МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ
им. М.В.ЛОМОНОСОВА

ТРУДЫ
ФАКУЛЬТЕТА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И
КИБЕРНЕТИКИ

№ 5

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

Под общей редакцией
чл.- корр. РАН Л.Н. Королева

Москва
2005

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению Редакционно-издательского совета
факультета вычислительной математики и кибернетики
МГУ им. М.В. Ломоносова

Программные системы и инструменты: Тематический сборник
П75 факультета ВМиК МГУ им. Ломоносова: № 5.
/ Под ред. Л.Н. Королева. – М: Издательский
отдел факультета ВМиК МГУ (лицензия ИД №05899 от
24.09.2001г.), 2005. - 201с.

В данный сборник включены научные работы и сообщения по темам, связанным с общими вопросами программирования и информатики, с проблемами эффективной эксплуатации IDS, со средствами визуализации, алгоритмами обработки образов, с проблемами распараллеливания программ и распределенных вычислений, а также связанные с проблемами и опытом внедрения современного математического программного обеспечения на платформе IBM Regatta. и другие сообщения, в частности посвященные сенсорным сетям.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2004 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

ISBN 5-89407-216-6

СОДЕРЖАНИЕ

---

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 02-01-00261
[2] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 02-07-90130

СБОРНИК

" Программные системы и инструменты"

От редколлегии:

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные в основном описаниям инструментальных программных систем, разработанных авторами публикаций.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Большинство статей представляют собой тексты докладов, прочитанных на Ломоносовских Чтениях 2004 на факультете ВМиК.

Перечень разделов сборника №5 таков:

- Общие вопросы программирования и информатики
- Средства визуализации и обработка образов
- Параллельные и распределенные вычисления
- Сенсорные сети
- Инструментальные средства и сообщения

Статьи сборника будут интересны специалистам, разрабатывающим прикладные программные системы и использующим новые информационные технологии.

Редколлегия

Раздел I
Общие вопросы программирования и информатики

Брусенцов Н.П.

Обобщение булевой алгебры

Булева алгебра в ее систематически упорядоченном виде [1] с базисными операциями конъюнкции, дизъюнкции и инверсии терминов представляет собой наиболее естественный и совершенный инструментарий классической (двухзначной) логики. Это алгебра четких и нечетких множеств терминов – элементарных конъюнкций, а также четких классов охарактеризованных совокупностями терминов вещей – дизъюнктивных нормальных форм. Четкость булевых классов обусловлена законом исключенного третьего, в силу которого отношение вещей к классам носит строго двухзначный характер: каждая из вещей рассматриваемого рода либо необходимо включена в данный класс, либо необходимо не включена в него (третьего не может быть). Этой четкости булева алгебра обязана простотой и замкнутостью – любая комбинация классов всегда есть класс.

Вместе с тем, исключенность третьего – причина того, что булева алгебра и классическая логика неадекватны содержательному отображению действительности, не вполне соответствуют интуиции, здравому смыслу. Убедительным подтверждением этого служит отношение следования, как известно, в булевой алгебре невыразимое, а вернее сказать, вырожденное в так называемую материальную импликацию, парадоксы которой логики безуспешно пытаются преодолеть. Сложность и запутанность проблемы содержательного следования в современной логике [2] объясняется тем, что в этой логике исключено то третье, без которого следование немыслимо.

В аристотелевой силлогистике непарадоксальное содержательное следование представлено общеутвердительной посылкой «Всякое x есть y» («y содержится в x», «из x следует y»). Не удивительно, что силлогистику не удается «погрузить» в современные логические исчисления. Дело не в том, будто бы она «узкая» система, несовместимая с понятием пустого множества, а в ее диалектической трехзначности [3], немыслимой при исключенности третьего.

Множественная (интенсиональная) формулировка отношения следования осуществима средствами булевой алгебры, пополненной префиксным дизъюнктом $\vee$ – *функтором существования* (принадлежности вещи представленной выражением совокупности

вещей). Так, общеутвердительная посылка «Всякое x есть y» моделируется [4] конъюнктивной совокупностью (множеством) вещей:

$$\mathrm{V}x\mathrm{V}'xy'\mathrm{V}y'$$

Этой совокупности необходимо принадлежат x-вещи и y'-вещи (не-y-вещи), а принадлежность xy'-вещей исключена, поэтому каждая x-вещь не может не быть y-вещью и каждая y'-вещь непременно есть x'-вещь. Другими словами, из x необходимо следует y, и в силу контрапозитивности следования, не-y не может не быть не-x.

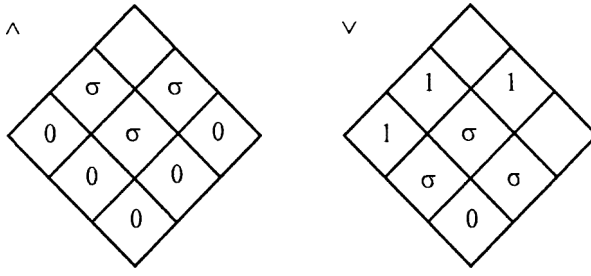
Существенная особенность данного выражения содержательного следования – упущенность (умолчание) в нем информации о принадлежности представленному им множеству $x'y$-вещей. В совершенной нормальной форме этого выражения – $\mathrm{V}xy\mathrm{V}'xy'\mathrm{V}x'y'$ явно обозначена принадлежность xy-вещей и $x'y'$-вещей, антипринадлежность xy'-вещей, тогда как $x'y$-вещи просто не упомянуты, принадлежность их не зафиксирована – не принадлежат и не антипринадлежат. Таким образом выражено третье (промежуточное) значение принадлежности, называемое Аристотелем *привходящим*. При этом сохраняется возможность вписать взамен умалчивания дизъюнкт $\mathrm{V}x'y$ либо его отрицание $\mathrm{V}'x'y$. В обоих случаях представляющее следование *нечеткое* множество вещей превращается в четкое множество, в первом случае $\mathrm{V}xy\mathrm{V}'xy'\mathrm{V}x'y\mathrm{V}x'y'$, представляющее следование, но не эквивалентность, а во втором случае $\mathrm{V}xy\mathrm{V}'xy'\mathrm{V}'x'y\mathrm{V}x'y'$, представляющее эквивалентность $x \leftrightarrow y$.

Аналогичное умалчиваемое третье имеет место в булевой алгебре элементарных конъюнкций и дизъюнкций, представляющих собой множества и классы первичных терминов, которыми обозначены несоставные особенности вещей. В рассуждении с n терминами n-арная конъюнкция (четкое их множество) именует конкретную вещь (индивид). Например, при n=3 конъюнкция $xy'z$ означает вещь, которой необходимо присущи особенности x и z, а присущность особенности y необходимо исключена (присуще не-y). Но принадлежность термина множеству может принимать и третье значение, выражающееся в том, что этот термин умалчивается в конъюнкции. Так, в том же рассуждении при n=3 конъюнкция xy' представляет *нечеткое множество* особенностей, означающее не конкретную вещь, а неединичный (неиндивидный) класс вещей: $xy' \equiv xy'z \vee xy'z'$. Умолчание термина придает ему привходящий статус – не присущ и не антиприсущ. Можно называть это «неопределенностью», но ясно, что третье, вопреки общепринятому закону логики, не исключено не только в действительности, а и в самой двухзначной булевой алгебре, правда, только на 1-ой ее ступени, в алгебре элементарных конъюнкций и дизъюнкций. Необходимое обобщение предполагает распространение аналогичной трехзначности на алгебру вещей, охарактеризованных

множествами терминов, т. е. на булеву алгебру в целом, что и было осуществлено выше применительно к интенсиональному (множественному) варианту булевой алгебры.

Современная же булева алгебра – система экстенсиональная (алгебра классов), поэтому непосредственное сообщение ей содержательного характера связано с приданием нечеткости классам вещей путем модификации дизъюнктивной нормальной формы. Требуется отделить привходящие вещи от необходимо исключенных. Это осуществимо в двух вариантах: 1) сохранив принятое ныне умалчивание как исключение с необходимостью, ввести функтор привходящего, например, в виде префикса σ, означающего $\sigma\upsilon\mu\beta\varepsilon\beta\eta\kappa o\varsigma$ – привходящее; 2) ввести функтор исключения с необходимостью, например, в виде надчеркивания либо знака «минус», а привходящее умалчивать. Содержательное следование $x \Rightarrow y$ в первом варианте выразится СДНФ $xy \vee \sigma x'y \vee x'y'$ и минимальной формой $x \vee \sigma x'y \vee y'$, а во втором варианте соответственно: $xy \vee \overline{xy'} \vee x'y'$ и $x \vee \overline{xy'} \vee y'$ либо $xy \vee \text{-}xy' \vee x'y'$ и $x \vee \text{-}xy' \vee y'$.

Первый вариант заслуживает предпочтения как совместимый с современной необобщенной алгеброй, не нарушающий традицию умалчивать исключенное, т. е. выражение четких классов вещей остается неизменным. Например, СДНФ и минимальное выражение материальной импликации $x \to y$ остаются теми же, что и в необобщенной алгебре: $xy \vee x'y \vee x'y'$ и $x' \vee y$. Замечательно, что при обощении по первому варианту остаются в силе принятые в булевой алгебре приемы преобразования выражений с учетом, естественно, того, что третье не исключено и находится в промежутке между крайними значениями, так что конъюнкция и дизъюнкция определены трехзначными таблицами Пирса:



Адекватность обобщенной алгебры содержательной логике убедительно подтверждается полноценным отображением в ней силлогистики Аристотеля, базисные суждения которой представляются трехзначными функциями двухзначных переменных:

$$Axy = xy \vee \sigma x'y \vee x'y', \quad Ixy = xy \vee \sigma x'y'.$$

Доказательство модуса Barbara: $AxyAyz \Rightarrow Axz$, принятого в силлогистике в качестве аксиомы, обобщенная алгебра осуществляет путем следующего тождественного преобразования исходной конъюнкции суждений:

$AxyAyz \equiv (xy \vee \sigma x'y \vee x'y')(yz \vee \sigma y'z \vee y'z') \equiv xzy \vee \sigma x'yz \vee \sigma x'y'z \vee x'y'z'$.

Элиминацией по Булю-Порецкому термина y полученное выражение редуцируется в $xz \vee \sigma x'z \vee x'z' \equiv Axz$.

Таким же образом доказывается справедливость «сомнительных» с точки зрения классической двухзначной логики модусов третьей и четвертой фигур. Более того, оказывается, что из посылок модуса Bamalip необходимо следует не только частное, но и общее заключение:

$AzyAxy \equiv (zy \vee \sigma z'y \vee z'y')(xy \vee \sigma x'y \vee x'y') \equiv xzy \vee \sigma xyz' \vee \sigma xy'z' \vee x'y'z'$,

что элиминацией термина y редуцируется в $xz \vee \sigma xz' \vee x'z' \equiv Azx$.

Литература

1. Брусенцов Н.П. Упорядочение булевой алгебры // Программные системы и инструменты. Тематический сборник № 3. Под редакцией чл.-корр. РАН Л.Н.Королева. – М.: Издательский отдел факультета ВМиК МГУ, 2002. С. 11-27.
2. Брусенцов Н.П. О сложности и запутанности проблемы логического следования // Современная логика: проблемы теории, истории и применения в науке. Материалы VIII общероссийской научной конференции 24-26 июня 2004 г. – СПб, 2004. С. 231-233.
3. Брусенцов Н.П. Трехзначная интерпретация силлогистики Аристотеля // Историко-математические исследования. Вторая серия. Вып. 8 (43). – М.: «Янус-К», 2003. С. 317-327.

Брусенцов Н.П. Искусство достоверного рассуждения. Неформальная реконструкция аристотелевой силлогистики и булевой математики мысли. – М.: Фонд «Новое тысячелетие», 1998. С. 84-108.

Брусенцов Н.П., Владимирова Ю.С.

Булевы уравнения и логический вывод

Джордж Буль усматривал в решении логических уравнений наиболее общую проблему логики [1, с. 87]: «...задано логическое уравнение, содержащее символы x, y, z, w. Требуется найти логически интерпретируемое выражение для выяснения отношения класса, обозначенного через w, к классам, обозначенным через x, y, z и т.д.» Он формулирует ту же проблему и более кратко: «Задано (логическое) уравнение; найти выражение одного термина в функции остальных» [2, с. 325]. Этот подход нашел понимание и получил дальнейшее развитие у Э. Шредера, У.С. Джевонса, П.С. Порецкого [3]. Однако в современной математической логике он не воспринят, и решение булева уравнения сведено теперь к выявлению удовлетворяющих этому уравнению индивидных конъюнкций (n-ок) терминов, что равносильно преобразованию его в СДНФ единичную форму. Таким образом, вместо установления обусловленных уравнением взаимосвязей между терминами определяется класс удовлетворяющих ему индивидных вещей («предметов»).

Развитию и компьютерной реализации решения логических уравнений в булевом понимании посвящены наши работы [4-7], в которых существенно усовершенствован метод Буля-Порецкого, предложено более простое решение, а также совокупностная интерпретация булевой алгебры классов, открывающая возможность алгебраического манипулирования булевыми выражениями как совокупностями индивидных конъюнкций терминов, преобразуемыми теоретико-множественными операциями пересечения, объединения и инверсии. Сами эти операции выполняются компьютером как побитные конъюнкция, дизъюнкция и отрицание битных векторов (так называемых «ДК-шкал»), которыми закодированы СДНФ-выражения.

В связи с обобщением булевой алгебры классов путем включения в нее понятия «нечеткий класс» [8] решение логических уравнений как способ получения умозаключений из определяемой ими взаимосвязанности терминов обретает новые возможности. В частности, булево исчисление классов оказывается распространимым на силлогистику Аристотеля – систему, наиболее адекватно отображающую естественноязыковое рассуждение, но упорно не вписывающуюся в исчисления «классической» логики. Представление суждений силлогистики в виде нечетких классов вещей, охарактеризованных содержащимися в суждениях терминами, позволяет алгебраически исследовать выраженные суждениями

взаимосвязи, причем использовать компьютер для получения искомых умозаключений.

Общеутвердительное силлогистическое суждение «Всякое x есть y», выражающее отношение содержательного логического следования y из x, отображается в обобщенной булевой алгебре нечетким классом вещей, охарактеризованных терминами-особенностями x и y [8]: $xy \vee \sigma x'y \vee x'y'$, где σ - символ, обозначающий третье промежуточное «значение истинности»: не «есть» и не «нет», не «1» и не «0», $0 < \sigma < 1$. В противоположность xy- и $x'y'$-вещам, необходимо включенным в данный класс, и умалчиваемым в выражении xy'-вещам, которые необходимо исключены, $\sigma x'y$-вещи имеют привходящий статус – не включены и не исключены с необходимостью (четко). Наличие $x'y$-вещи не утверждает и не отрицает следования y из x, оно несущественно для этого отношения. В булевой алгебре четких классов отношение следования вырождается в материальную импликацию $xy \vee x'y \vee x'y'$ с присущими ей «парадоксами»: на x'-вещах она удовлетворяется независимо от y, а на y-вещах независимо от x. В результате «классическая» логика оказывается бессодержательной, лишенной здравого смысла.

СДНФ-выражения нечетких классов вещей представляют собой нечеткие дизъюнктивные совокупности индивидных конъюнкций терминов, отобразимые ДК-шкалами тритов, подобно тому как СДНФ-выражения четких классов отображаются ДК-шкалами битов [7]. Так, выражение $xy \vee \sigma x'y \vee x'y'$ однозначно кодируется четырехтритной шкалой (вектором тритов) +-0+, в которой "-" означает исключенность xy'-вещей, а "0" – привходящий статус $x'y$-вещей. В трехтерминном x,y,z-универсуме тот же нечеткий класс выразится шестичленной СДНФ $xyz \vee xyz' \vee \sigma x'yz \vee \sigma x'yz' \vee x'y'z \vee x'y'z'$ и соответственно восьмитритным кодом ++--00++. Суждение «Всякое y есть z» отображается нечетким классом $yz \vee \sigma y'z \vee y'z'$ и восьмитритным кодом +-0++-0+.

Теперь вывод заключения из посылок «Всякое x есть y» и «Всякое y есть z» (совершенный модус Barbara, принимаемый Аристотелем в качестве аксиомы) реализуется компьютером путем пересечения, т. е. потритной конъюнкции, соответствующих ДК-шкал:

++--00++ ⋂ +-0++-0+ = +---0-0+.

Декодируя шкалу-результат, получим $xyz \vee \sigma x'yz \vee \sigma x'y'z \vee x'y'z'$, что элиминацией термина y сводится к $xz \vee \sigma x'z \vee x'z'$ - «Всякое x есть z».

Решение относительно термина y уравнения $xy \vee \sigma x'y \vee x'y' = 1$, выражающего содержательное следование y из x, по формуле $y = ye \vee \neg y \neg e$ [7] при помощи ДК-шкал: y = +-+-, $\neg y$ = -+-+, e = +-0+, $\neg e$ = -+0- дает в результате: (+-+- ⋂ +-0+) ⋃ (-+-+ ⋂ -+0-) =

= +-0- ⋃ -+-- = ++0-. Декодируя полученную шкалу, имеем $y = x \vee \sigma x'y$, что в полном соответствии с содержательным следованием y из x.

Литература

1. Boole G. An investigation of the laws of thought on which are founded the mathematical theories of logic and probabilities. London, 1854.
2. Стяжкин Н.И. Формирование математической логики. – М.: «Наука», 1967.
3. Порецкий П.С. О способах решения логических равенств и об обратном способе математической логики. Опыт построения полной и общедоступной теории умозаключений над качественными формами. – Казань, 1884.
4. Брусенцов Н.П. Начала информатики. – М.: Фонд «Новое тысячелетие», 1994.
5. Владимирова Ю.С. Конструктная реализация булевой алгебры.// Интегрированная система обучения, конструирования программ и разработки дидактических материалов. – М.: Изд-во ф-та ВМиК, 1996, с.44-69.
6. Brusentsov N.P., Vladimirova Yu.S., Solution of Boolean Equations. // Computational mathematics and modeling, Vol. 9, № 4, 1998, pp. 287-295.
7. Брусенцов Н. П., Владимирова Ю. С. Компьютеризация булевой алгебры // Доклады Академии Наук, 2004, № 395, т. 2, с. 7-10.
8. Брусенцов Н.П. Обобщение булевой алгебры // В этом сборнике. С.

Гуляев А.В. Зарецкий С.В.

Проблема эффективной эксплуатации IDS

Преимущества систем обнаружения вторжений

Повысить уровень защищенности корпоративной сети можно с помощью средств обнаружения вторжений (Intrusion Detection). Средства обнаружения вторжений хорошо дополняют защитные функции межсетевых экранов. Если межсетевые экраны стараются отсечь потенциально опасный трафик и не пропустить его в защищаемые сегменты, то средства обнаружения вторжений анализируют результирующий, (прошедший через межсетевой экран или созданный внутренними источниками) трафик в защищаемых сегментах и выявляют атаки на ресурсы сети или действия, которые могут классифицироваться как потенциально опасные (подозрительные). Средства обнаружения вторжений могут также использоваться и в незащищенных сегментах, например, перед межсетевым экраном, для получения общей картины об атаках, которым подвергается сеть извне.

Средства обнаружения вторжений автоматизируют такие необходимые элементы деятельности администратора системы безопасности, как:

- регулярный анализ событий, связанных с доступом к ресурсам, по данным журналов аудита
- выявление атак и подозрительной активности,
- выполнение ответных действий - реконфигурация средств защиты (межсетевых экранов, настроек операционных систем и т.п.) для пресечения подобных атак в будущем.

Почему стоит выделить системы обнаружения вторжений среди средств обеспечения безопасности? Рассмотрим причины, по которым системы обнаружения вторжений имеют преимущество над другими средствами обеспечения безопасности.

Надо учитывать, что технологии основных средств безопасности, а именно: межсетевых экранов, централизованной системы аутентификации, разграничения доступа основаны на стандартах, написанных в военных ведомствах США в 70-80 годах 20-го века. Согласно моделям, заложенным в этих стандартах, субъекту (пользователю, программе, сетевому пакету) разрешается доступ при

предъявлении уникального, присущего только ему элемента. Как правило, этот элемент- пароль. Уникальность этого элемента- слабое звено в этой модели. При современных методах получить доступ к паролю или ключу не вызовет никаких проблем, особенно с учетом развития глобальных сетей.

- Этот недостаток в модели безопасности вызван спецификой военных организаций, где его разрабатывали. Там практически нет «чужих».
- Другой проблемой в модели является неготовность ее к постоянно возрастающему числу уязвимостей в операционных системах и программном обеспечении.

Межсетевые экраны, не смотря на то, что считаются самыми надежными средствами обеспечения безопасности, по сути своей мало приспособлены для обнаружения атак. Атаки с помощью тунеллирования через другие протоколы, подмена адреса, неправильная конфигурация экрана, возможность атаки самого экрана, часто существующие возможности входа в сеть помимо канала, защищенного межсетевым экраном – все это снижает доверие к технологии межсетевых экранов, как к технологии обеспечения безопасности.

Приведем цифры статистики. В рамках исследования, которое финансировало Министерство обороны США, группа экспертов провели анализ защищенности 8932 военных информационных систем. В 88% случаев проникновение было успешным. Администраторы только 390 из этих систем обнаружили атаки и всего 19 из них сообщили о нападениях.

Т.е. только в 5% систем смогли зафиксировать атаки и только в 0.24% их обнаружить. Причина кроется в том, что традиционные средства защиты, такие как: разграничение доступа, фильтрация трафика, аутентификация разрабатывались без учета современных аспектов технологий вторжений. Решение проблемы - современные механизмы обнаружения атак.

Можно сделать вывод о том, что система обнаружения вторжений – самое эффективное средство для обеспечения безопасности в сети предприятия.

Технологии обнаружения вторжений

Рассмотрим два основных подхода обнаружению атак: статистический и экспертный:

Статистический подход используется при обнаружении аномального поведения. Отклонение от среднего значения (т.е. дисперсия) профиля нормального поведения и дает сигнал

администратору о том, что зафиксирована атака. Средние частоты и величины переменных вычисляются для каждого типа нормального поведения (например, количество входов в систему, количество отказов в доступе). О возможных атаках сообщается, когда наблюдаемые значения выпадают из нормального диапазона, т.е. превышают заданный порог.

Параметры, которые включаются в шаблон поведения объекта, могут быть отнесены к следующим группам:

- Числовые параметры (количество переданных данных по различным протоколам, загрузка цп, число файлов к которым осуществлялся доступ и т.п.)
- Категориальные параметры (имена файлов, команды пользователя, открытые порты и т.д.)
- Параметры активности (число обращений к файлам или соединений за единицу времени и т.д.)

Преимущества статистических систем:

- Могут обнаруживать неизвестные атаки
- Позволяют обнаруживать более сложные атаки, чем другие методы.

Недостатки статистических систем:

- Трудность задания порогового значения
- Систему можно обмануть постепенным изменением режима работы
- Более высокая вероятность получения ложного сообщения об атаке, чем в других методах
- Статистические методы не очень корректно обрабатывают изменения в деятельности пользователя. В результате могут появляться как ложные сообщения, так и пропущенные атаки.
- Статистические методы не могут обнаружить атаки со стороны субъектов, для которых невозможно описать шаблон типичного поведения.
- Статистические методы не справляются с обнаружением атак со стороны субъектов, которые с самого начала выполняют несанкционированные действия.
- Нечувствительны к порядку следования событий

- Требуют предварительной настройки (задания пороговых значений)

Экспертные системы работают на основе правил. Правила описывают сценарии атак. Экспертная система- это система, которая в контексте обнаружения атак принимает решение о принадлежности того или иного события к классу атак на основании имеющихся правил. Правила основаны на опыте специалистов и хранятся в специальном хранилище. В большинстве случаев экспертные системы опираются на так называемые сигнатуры, которые и ищутся в контролируемом пространстве.

Сигнатуры- это шаблоны, сопоставляемые известным атакам или злоупотреблениям в системах. Сигнатуры могут быть простыми (строка знаков, соответствующая поиску определенных условий или команд), либо сложными (изменение состояния защиты, описанное формальным математическим выражением, последовательностью действий или совокупностью строк журнала регистрации).

Достоинства экспертных систем:

- Простота реализации
- Скорость функционирования
- Отсутствие ложных тревог

Недостатки экспертных систем:

- Неспособность к обнаружению неизвестных атак (требуется поддержка базы сигнатур в актуальном состоянии)
- Небольшие изменения в атаке приводят к невозможности ее обнаружения
- Система зависит от квалификации специалистов, заполняющих базу знаний

Современные системы обнаружения атак используют примерно следующее соотношение статистических и экспертных методов: 30% и 70%.

Проблемы в системах обнаружения вторжений

Несмотря на описанные преимущества, системы обнаружения атак не являются универсальным решением. Они, как практически и все другие средства имеют свои ограничения.

Основные проблемы в IDS:

- Рост квалификации злоумышленников, эффективности автоматизированных хакерских средств и их многообразие

- Для сигнатурных IDS: уязвимости появляются быстрее, чем средства, обнаруживающие их использование. Необходимо постоянно обновлять базы сигнатур.

- отсутствует единый стандарт для сетевых протоколов; существующие стандарты не описывают все возможные состояния сетевого взаимодействия, и разработчики вольны выбирать те решения поставленной задачи, которые покажутся им оправданными; соответственно такое положение дел порождает ошибки, внесенные на стадии проектирования

- возможно неправильное конфигурирование IDS во время установки или администрирования, т.е. ошибки на стадии эксплуатации;

- как и любые программные продукты, IDS содержит ошибки, внесенные на стадии проектирования, разработки или реализации.

Большая вероятность ошибки в ПО IDS приводит к различным проблемам.

Например, изменения в данных, передаваемых в сеть, могут позволить обойти IDS. Передаваемая по сети информация может быть по-разному обработана IDS и атакуемой системой, а из-за различий в стеках протоколов появляется возможность создать последовательность пакетов, которая будет принята IDS, но отвергнута атакуемой системой. В таком случае IDS не знает, что целевая система не приняла пакет, и атакующий может воспользоваться этим для сокрытия факта проведения атаки, посылая специально созданные пакеты. Создавая ситуацию, когда целевая система отбрасывает пакеты, а система обнаружения атак их принимает, нарушитель как бы «вставляет» данные в анализатор событий IDS. Таким образом, даже вторжение с известной сигнатурой может остаться незамеченной IDS.

Опыты подтверждают, что небольшие модификации известных вторжений дают возможность злоумышленникам обходить системы обнаружения вторжений.

Рассмотрим возможные изменения атаки, которые могут помочь обойти IDS:

- Изменение последовательности команд
  Пример: Измените стандартную последовательность действий при атаке. Например, если атака одинаково осуществляется и при 'a;b;c' и при 'b;a;c', то большинство систем обнаружения атак обнаружит первый вариант реализации атаки, но не увидит ее вариации.
- Распределение по хостам команд
  - Разделение атаки между несколькими удаленными IP-

адресами/системами. Например, с узлов X и Y запускаем 'a' с X, 'b' с Y, а затем 'c' опять с X.

- Распределенное по времени сканирование
  Дополнительная атака - Осуществление атаки очень медленно. Т.к. размеры буфера для хранения регистрационных данных ограничены, то первые команды для атаки могут быть потеряны к тому моменты, когда реализуются последние команды атаки.
- Распределенное по хостам сканирование
- Обработка большого объема трафика
- Модификация сигнатуры атак. Используя различные реализации стека сетевых протоколов можно обойти IDS:
  Например: изменение контрольной суммы в ip пакете, внесение некорректной версии протокола IP и т.д.

Вывод

Таким образом, существует ряд проблем, связанных с возможностью обнаружения атак в IDS:

- Потребность в постоянном контроле обновления баз сигнатур
- Возможность обхода IDS за счет изменения вторжений
- Возможность обхода IDS за счет распределения вторжений

Без учета этих факторов невозможна эффективная эксплуатация системы обнаружения вторжений. Администратору нужно следить за обновлениями, проверять возможность обнаружения IDS измененных вторжений.

Пути решения задачи

Решением описанных проблем может служить методика тестирования, средств обнаружения вторжений, обеспечивающая следующие функции:

- Контроль обновления сигнатур с помощью тестирования с использованием готовых вторжений и обновлений сигнатур
- Тестирование IDS с помощью измененных и распределенных вторжений.

Суть тестирования заключается в имитации действий злоумышленника с помощью автоматизированной системы.

Литература

1. Лукацкий А.В. Обнаружение Атак. БХВ-Петербург. 2003 г

2. Вакка Дж. Секреты безопасности в Internet. К.: Диалектика, 1997
3. Зегжда Д.П Ивашко А.М. Как построить защищенную информационную систему. Санкт Петербург 1997
4. Ален Хаусхолдер, Кевин Хаул, Чад Дугерти Компьютерные атаки угрожают безопасности Интернет. Открытые системы №7-8,2002
5. Peter Mell. Computer Attacks: What They Are and How to Defend Against Them. NIST, Computer Security Division, 1999
6. Richard Power. Current and Future Danger: A CSI Primer on Computer Crime and Information Warfare. Computer Security Institute. 1995

Раздел II.
Средства визуализации и обработка образов

Королев Л.Н., Мещеряков Д.К.

Исследование возможностей предобработки речевых сигналов и использования генетического алгоритма в задаче нормализации речевых образов[2]

Введение

В задачах идентификации образов часто приходится иметь дело с необработанными данными, полученными оцифровкой тех или иных аналоговых материалов. Очевидно, необходима предварительная нормализация этих данных для приведения их к единому образцу, т.е. некое преобразование, которое позволяло бы абстрагироваться от искажений, свойственных оцифровке, – шума, неточной воспроизводимости образцов, различий в настройках оборудования. Один из таких методов ([1]) предполагается использовать для распознавания слов в речевом сигнале, а для этого использовать вместо самого сигнала данные о его рельефе, поскольку именно они наиболее содержательно описывают содержащиеся в речевом сигнале слова. Данная работа частично исследует применимость этого подхода для решения задачи идентификации слов.

Общая постановка задачи

Имеются две сеточные функции $f(x_i)$ и $s(x_i)$. Необходимо выяснить, отвечают ли эти две сеточные функции одному и тому же в заданном смысле сигналу. Исследуемый подход предполагает подвергнуть $f(x_i)$ деформации с помощью многопараметрического преобразования заданного вида, получив $f^*(x_i)$, а затем сравнить $f^*(x_i)$ и $s(x_i)$ при помощи выбранной нормы. В зависимости от значения нормы разности между $f^*(x_i)$ и $s(x_i)$ можно будет принять решение, отвечают ли эти функции одинаковым сигналам.

Деформирующее преобразование предполагается брать в виде

$$f^*(x_i)=\lambda f(\mu(x_i)) \quad (1)$$

[2] Работа выполнена при поддержке гранта РФФИ № 02-07-90130.

где λ - положительное вещественное число и $\mu(x_i)$ – функция, задающая неубывающую ломаную.

В рамках данного исследования проверялась применимость подхода к сравнению речевых сигналов, содержащих произнесения слов. Преобразование в предложенном виде позволило бы абстрагироваться от различий в уровне сигнала в разных его образцах, вызванных различиями в условиях записи сигнала и другими условиями, не влияющими на априорную тождественность образцов, и от различий в скорости записи сигнала. Последнее очень существенно, поскольку одно и то же слово можно произносить с разной скоростью, с паузами или растяжениями, и от всего этого необходимо абстрагироваться при сравнении.

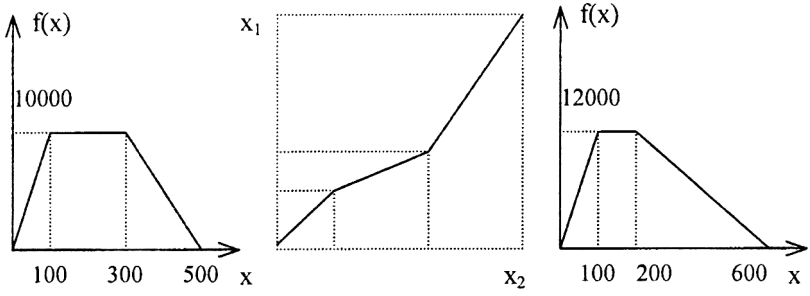


Рис. 1

На рисунке 1 показаны два сигнала и примерный вид преобразования, приводящего один из сигналов к другому.

В реальных приложениях число узлов ломаной может быть очень велико, и тогда задача выбора их координат становится очень сложной. Для решения этой задачи предлагается использовать генетический алгоритм – параметры преобразования сопоставить признакам особей, а в качестве функции качества использовать некоторую норму разности между сигналами (см. [1] и [4]).

При условии, что есть возможность строить такие преобразования для любых пар сходных по смыслу сигналов, методику можно использовать для распознавания слов следующим образом: выбрать большое число различных произнесений каждого из распознаваемых слов, для каждого слова выбрать некоторый эталонный вариант его произнесения и построить наборы параметров, с которыми все остальные варианты произнесения могут быть преобразованы к эталону. Теперь любое исследуемое слово можно по очереди сравнивать со всеми вариантами произнесения каждого известного системе слова. Как только исследуемое слово совпадает (достаточно близко по норме) с каким-то вариантом известного слова, можно считать, что исследуется вариант этого известного слова. Если исследуемое слово не совпадает

ни с одним вариантом ни для одного известного слова, можно считать, что это слово системе неизвестно.

Конкретная постановка задачи и предполагаемое решение

Имеются два импульсно-кодированных сигнала $f(x_i)$ и $s(x_i)$, содержащих произнесение одного и того же слова. Необходимо с помощью генетического алгоритма построить набор параметров для вышеозначенного преобразования (1), который бы обеспечивал максимальную близость этих сигналов по норме

$$\Sigma(\lambda f(\mu(x_i))-s(x_i))^2$$

Сигналы записаны в условиях изолированного помещения и содержат шум с очень низким уровнем, поэтому задача шумоподавления не ставится. Имеющийся опыт позволяет судить, что все равно не следует использовать сигнал в исходном виде, поскольку быстроосциллирующий характер сигнала препятствует устойчивой сходимости генетического алгоритма – сигнал содержит слишком много информации, не относящейся к произнесению слова, но характеризующей особенности голоса произносящего.

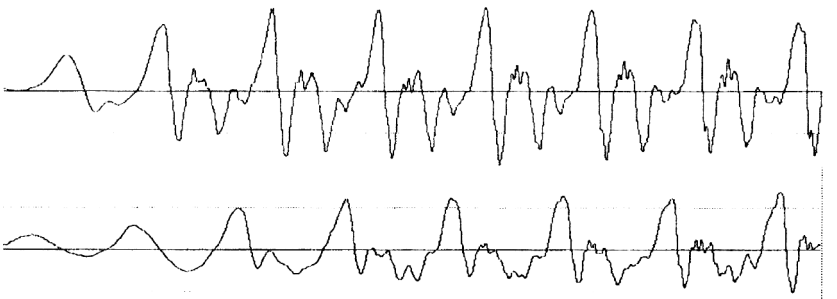


Рис. 2

На рисунке 2 приведены две осциллограммы, соответствующие началам произнесений одного и того же слова одним и тем же человеком. Хотя некоторое сходство в форме сигналов усматривается, едва ли можно что-либо сказать наверняка. Очевидно, сравнивать сигналы по точкам неэффективно. В связи с этим предлагается использовать вместо сигналов их огибающие, например, верхние. Однако уже накопленный опыт позволяет судить, что столь прямолинейное применение исследуемого подхода не дает каких-либо результатов (подробнее причины этого явления обсуждаются в экспериментальной части). Поэтому данное исследование посвящено поиску методов обработки сигналов, которые позволили бы выделить «словесную» составляющую

сигнала и абстрагироваться от «голосовой», т.е. деперсонализировать образцы. Существенно, что нельзя просто уменьшить частоту дискретизации ниже 8 кГц - в противном случае проявится так называемый алиасинг, и сигнал исказится существенно для решения задачи.

Практическая часть

В рамках исследования были разработаны и реализованы два вспомогательных программных средства. Программа SignalCruncher – основная программа вычислительного эксперимента - гипотетически позволяет подбирать параметры преобразования с помощью генетического алгоритма. Она существенно использует свободно распространяемый компонент GeneBase ([3]). Интерфейс программы позволяет выбирать число узлов (число точек ломаной) преобразования, число особей в популяции, число итераций генетического алгоритма, указывать, следует ли варьировать параметр λ или использовать значение, равное единице. После того, как выбраны значения всех настроек, пользователь может выбрать два файла формата PCM 16 bit mono ([5]) (частота дискретизации умышленно не проверяется), программа выполнит выбранное число итераций генетического алгоритма. На каждой итерации выводится значение функции качества для лучшей особи, время вычислений и изображение ломаной с набором параметров из лучшей особи. После завершения последней итерации полученный параметры сохраняются в файл, имя которого выбирается автоматически по несущественным для исследования правилам.

Функция качества вычисляется как разность по среднеквадратичной норме между неизмененным «эталонным» и деформированным «проверяемым» сигналом. Поскольку существенно, чтобы кривая преобразования была неубывающей, координаты узлов после их извлечения из машинного представления особи сортируются по возрастанию.

Программа Blur выполняет преобразование сигналов, аналогичное по сути размытию изображений, но реализованное для двумерных сигналов. Ядро свертки для преобразования выбиралось опытным путем, в конце концов выбор был остановлен на ядре (1/21, 2/21, 4/21, 7/21, 4/21, 2/21, 1/21), но позже это направление было оставлено.

Эксперимент и результаты

Для эксперимента использовалась коллекция сигналов, состоявшая из большого числа произнесений слова «пароль» на русском

языке несколькими людьми и нескольких реплик, не содержавших произнесения слова «пароль». Соответственно исследовались возможности приводить одно к другому произнесения одного и того же слова одним и тем же человеком, произнесения одного и того же слова разными людьми и произнесения разных реплик.

На начальном этапе производился поиск преобразования, которое позволило бы выделить из сигнала голосовую составляющую и оставить только речевую. Для этого использовался графический эквалайзер из пробной версии звукового редактора Sound Forge ([2]). Из априорных соображений выделялись частоты от примерно 100 Гц до примерно 600 Гц, а остальные подавлялись. Конкретные значения этих порогов варьировались в достаточно широких диапазонах. Эти эксперименты не дали какого-либо интересного результата. В некоторых случаях речь становилась неразборчивой на слух, но сигнал все равно содержал большое число быстрых осцилляций. Не удавалось подобрать параметры фильтра так, чтобы речь была разборчивой, но при этом сигнал был относительно гладким.

На рисунке 3 показан один из примеров. Вверху – исходный «сырой» сигнал с микрофона, внизу – результат применения фильтра, пропускающего частоты от 190 до 550 Гц и подавляющего остальные.

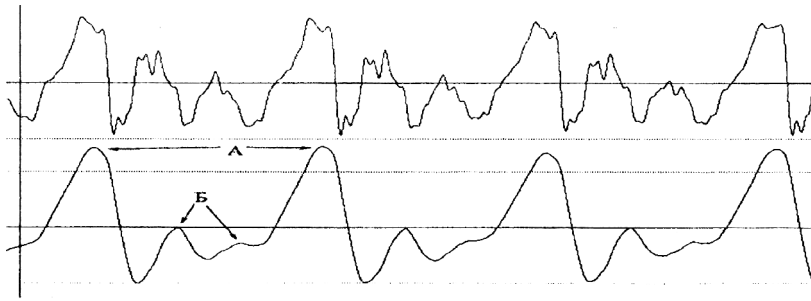


Рис. 3

Вертикальная линия в левой части рисунка – курсор звукового редактора, соответствует отсчетам с одним и тем же номером до и после преобразования. Новый сигнал, бесспорно, более гладкий, чем исходный, но в нем много участков, обозначенных на рисунке как «Б» (локальные экстремумы с небольшими по модулю значениями, расположенные периодически между локальными экстремумами с большими по модулю значениями). Из-за присутствия таких участков огибающая будет содержать большое число перепадов, а это сделает исследуемый подход трудно применимым для решения задачи. В идеале требуется оставить только участки, обозначенные как «А», т.е. превратить сигнал практически в синусоидальный, а участки «Б» как-либо замаскировать.

Практически на таких наборах данных невозможно эффективно различать сигналы – тестовые запуски показывают практически одинаковую сходимость генетического алгоритма по выбранной для эксперимента среднеквадратичной норме. Не удается уверенно различать реплики.

Следующим этапом было применение фильтра размытия. Этот опыт также не дал интересных результатов – участки «Б» сохранялись, а разборчивость речи в значительной степени терялась, и генетический алгоритм сходился одинаково как в случае одинаковых, так и в случае различных реплик. Поскольку экспериментировать с ядрами свертки затруднительно и очень сложно предсказать влияние выбора ядра на конечный результат, то это направление было оставлено.

Далее было решено сосредоточиться на возможностях программы Sound Forge и присутствующем в ней графическом эквалайзере. Графический эквалайзер позволяет задавать параметры фильтра различными способами. На ранних этапах эксперимента использовался режим «конверта», в котором параметры задаются с помощью кривой. Поскольку даже для кривой, изображающей прямоугольный импульс, левую и правую границу можно перемещать в широких пределах и с малым шагом, то перебор этих параметров без каких-либо априорных данных о параметрах фильтра бесперспективен. Поэтому было решено использовать задание параметров в «20 – полосовом» режиме. В этом режиме задаются коэффициенты усиления для 20 «центральных» частот, а коэффициенты остальных частот вычисляются программой Sound Forge автоматически. По очереди усиливалась одна из «центральных» частот, а остальные подавлялись. Каждый раз визуально исследовалась осциллограмма полученного сигнала, а сигнал исследовался на слух. Оказалось, что выделение «центральной» частоты 225 Гц обеспечивает получение почти синусоидального сигнала практически без участков «Б», в котором устойчиво различается речь. При этом результат не зависел от конкретного произносящего.

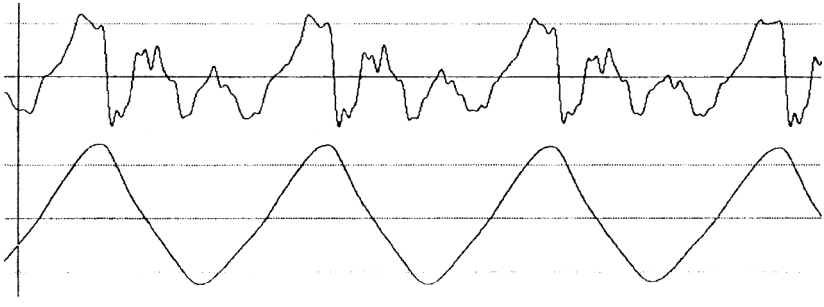


Рис. 4

На рисунке 4 показаны осциллограммы исходного и преобразованного сигналов. Вертикальная линия (курсор редактора) соответствует одним и тем же номерам отсчетов. При том, что сигнал превратился практически в синусоидальный, речь вполне различается. Одновременно сохраняется возможность отличить на слух женский голос от мужского. Если теперь передискретизовать сигнал с меньшей частотой (не ниже 8 кГц), то распознаваемость речи на слух сохранится а сигнал станет намного короче, что очень удобно для ускорения вычислений.

Тем не менее оказалось, что даже после такого преобразования сигналы все равно не распознаются устойчиво – вновь генетический алгоритм сходится примерно одинаково вне зависимости от априорного сходства сигналов. Тогда эксперимент был направлен на исследование получаемых с помощью генетического алгоритма результатов. Число узлов кривой было уменьшено до нескольких штук (трех-пяти). Обнаружилось, что в последовательных запусках при одних и тех же начальных данных получаются совершенно разные результаты. На каждой итерации значения координат узлов меняются практически произвольно и скачкообразно. В принципе, это вполне объяснимо, поскольку генетические алгоритмы при работе существенно используют псевдослучайные числа. В начале работы устанавливается некое начальное значение, а затем специальная функция по мере необходимости получает новые значения. Для того, чтобы последовательность чисел менялась от запуска к запуску, начальное значение выбирают, например, как функцию от текущего времени. Итого мы должны получать некую сучайность в работе генетического алгоритма, которая должна компенсироваться его сходимостью. Однако по той или иной причине в данном эксперименте уверенной сходимости генетического алгоритма не наблюдается. Это может быть связано с неудачной реализацией алгоритма или с неудачной реализацией функции качества, приведшей к накоплению погрешностей.

Заключение

Опытным путем найдено преобразование, приводящее речевой сигнал к почти синусоидальной форме и позволяющее в значительной степени деперсонализировать речевой сигнал. Это преобразование заметно сокращает число узлов в огибающей сигнала, упрощает выделение рельефа сигнала, практически полностью избавляет сигнал от интонации произношения, т.е. позволяет заметно продвинуться по пути превращения речевого сигнала именно в представление слова. В то же время обнаружено, что генетический алгоритм в выбранной реализации не вполне пригоден для подбора параметров

нормализующего преобразования речевых сигналов, что оставляет простор для дальнейших исследований в данном направлении.

Литература

1. Королев Л.Н. Некоторые методы «нормализации образов». Вестник МГУ. М., 2003.
2. Sound Forge and Sound Forge Studio Online Help. http://mediasoftware.sonymusic.com
3. Электронная документация к компоненту GeneBase. http://www.basegroup.ru
4. Стариков А. Генетические алгоритмы – математический аппарат. http://www.basegroup.ru
5. Кенцл Т. Форматы файлов Internet / Перев. с англ. – СПб: Питер, 1997. Стр. 256 – 259.

Тихонов А.В.

Среда визуализации функционирования распределенных программ

Введение

Сегодняшние информационные технологии постоянно имеют дело со значительными объемами информации. Способность человека визуально распознавать и интерпретировать сложные шаблоны, проистекающая из ведущего положения зрения среди всех чувств, является крайне важным дополнением к аналитическим методикам, которое позволяет производить качественную оценку данных. Поэтому, визуализация является важным средством для понимания явлений, описанных с помощью данных большого объема, полученных в результате моделирования или с помощью физического оборудования. Она, зачастую, позволяет исследователю подметить некоторую сложную закономерность или наоборот обнаружить аномалию в рассматриваемых данных.

В настоящей работе приводятся результаты разработки среды визуализации функционирования распределенных программ и примеры её практического использования.

1. Особенности визуализации распределенных программ

Говоря о визуализации функционирования распределенных программ, мы будем иметь в виду визуализацию, где в качестве объекта выступает некий программный или программно-аппаратный комплекс, содержащий некоторое количество самостоятельных взаимодействующих компонент, и, таким образом, задачей визуализации является доступное отображения структуры исследуемого распределенной программы, процесса его функционирования (последовательности действий, предпринимаемых отдельными компонентами), а также ключевых данных и кода распределенной программы.

Из-за большого числа состояний распределенной системы и сложных механизмов взаимодействия между отдельными элементами внутри системы бывает трудно достичь достаточного понимания работы системы, опираясь только на текстовый поток вывода или работая с несколькими такими потоками. Отладка и тестирование распределенной программы также представляют

непростую задачу. С расширением имеющихся текстовых средств графическими представлениями состояний программы, указанием связей, историей и другими элементами появляется возможность упростить понимание поведения распределенных программ.

Таким образом, можно говорить о ряде причин необходимости применения различных методов визуализации при работе с распределенными системами:

- Большая сложность рассматриваемых систем, в том числе большие объемы информации.
- Работа с абстрактными объектами, желание заменить их на интуитивно-понятные визуальные представления.
- Слишком детальный уровень представления информации, потребность в механизмах регулирования уровня детальности.
- Неочевидность алгоритмов распределенных программ, проблемы с их непосредственным восприятием, анализом и синтезом.

Типичным способом сбора и накопления информации о работе распределенной системы для последующего ее анализа является построение *трассы работы системы* – упорядоченного по времени списка событий и состояний, имевших место в различных компонентах распределенной системы [1].

Другим источником информации о распределенной системе является *файл описания структуры системы* – файл, содержащий иерархию компонентов и описание связей между ними, а также список допустимых типов событий, состояний и список параметров системы.

2. Требования к разрабатываемой системе

Изучение предметной области, истории распределенных вычислений и методов их визуализации, анализ практических потребностей пользователей и разработчиков распределенных программ, а также анализ современных пакетов визуализации (в т.ч. Pavane[2], Narcissus[3], LEADS[2] и др.) позволяют сформулировать следующие основные требования к реализации:

- Пакет визуализации не должен накладывать ограничений на разработчика программы, предназначенной для визуализации, то есть не должен требовать изменения исходных текстов программ, при разработке которых не учитывались потребности визуализатора.
- Стиль представления информации должен быть максимально очевидным для человека, незнакомого со средством визуализации, но знакомого с предметной областью.

- При разработке должны быть предусмотрены механизмы пользовательской настройки средства, позволяющие ему получить максимально удобный инструмент:
  - Изменение уровня детальности отображения информации – нужно предоставить пользователю механизм, позволяющий выбрать для визуализации нужные объекты с необходимой детальностью[4].
  - Изменение состава и порядка выводимых объектов.
- Среда должна предоставлять удобные средства навигации:
  - Навигация по времени работы программы,
  - Поиск определенных участков,
  - Навигация по степени детальности визуализации,
  - Навигация по отображаемым компонентам.
- Должна быть предусмотрена тесная интеграция с исходным кодом визуализируемой программы, позволяющая сопоставлять отдельные элементы и участки отображения с участками кода распределенная программы.
- При разработке должна учитываться возможность последующего переноса среды визуализации под другие платформы и графические библиотеки.

3. Архитектура среды визуализации

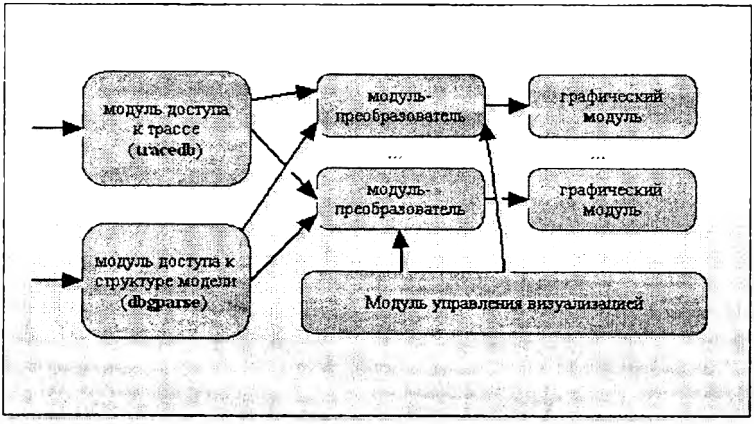


Рис. 1

Предлагаемая архитектура среды визуализации показана на Рис.1. Среда состоит из следующих компонентов:

Модуль доступа к структуре (dbgparse)

Данный модуль получает на вход описание структуры (например, в виде файла), содержащее информацию о компонентах распределенной программы и связях между ними. Он предоставляет унифицированный интерфейс к описаниям разных распределенных программ.

Модуль доступа к трассе (tracedb)

Этот модуль отвечает за унифицированный доступ к трассе работы системы – порядоченного по времени списка событий и состояний, имевших место в различных компонентах распределенной системы.

Использование модулей tracedb и dbgparse позволяет абстрагироваться от формата хранения данных при написании модулей-преобразователей и графических модулей. Это дает возможность построить расширяемую среду визуализации, работающую с различными форматами входных данных, и увеличивать количество поддерживаемых форматов без модификации основного кода.

Модуль управления визуализатором

Отвечает за общее управление визуализатором. Занимается отображением виджетов, позволяющих пользователю управлять масками интересующих его сущностей, временным интервалом, детализацией структуры компонентов. При появлении изменений уведомляет о них модули-преобразователи.

Модули-преобразователи

Отвечают за получение информации из трасс через интерфейс модуля tracedb и подготовку её к отрисовке специфичным для конкретного модуля образом. Полученная информация передается для отображения в соответствующий графический модуль.

Примерами модулей-преобразователей могут являться модули, осуществляющие рассчет графиков значений переменных и параметров

распределенной программы, диаграммы линий жизни её процессов и т.п.

Графические модули

Отвечают за отрисовку (с помощью стандартных виджетов или специально разработанных для данного вида отображения) данных, полученных от соответствующего модуля-преобразователя.

4. Описание построенной среды визуализации распределенных программ

В рамках описанной выше архитектуры была создана среда визуализации, позволяющая:

- изучать трассы работы распределенных систем с помощью нескольких видов отображения;
- экспортировать результаты визуализации во внешние приложения и/или на принтер;
- настраивать среду для работы с различными форматами входных данных (что полезно при подключении среды визуализации к уже существующим практическим проектам).

В состав созданной среды вошли следующие инструменты:

- Визуализатор трассы работы распределенной системы.
- Анализатор производительности распределенной системы.
- Инструменты для работы с трассой и структурой системы.

4.1. Визуализатор трассы работы распределенной системы

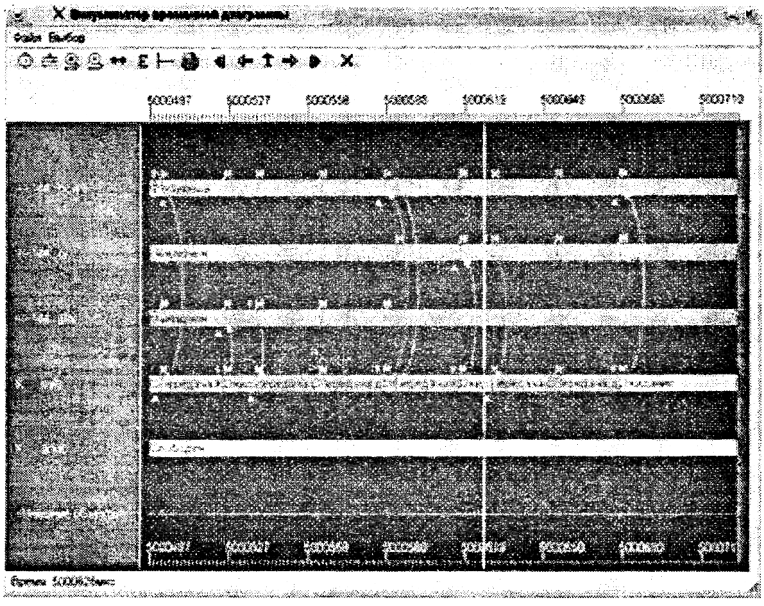
Визуализатор предназначен для наглядного отображения трассы работы распределенной системы, позволяющего качественно ее проанализировать, оценить загруженность отдельных компонентов, последовательности их состояний и взаимодействий между ними.

Визуализатор, построенный в рамках описанной выше архитектуры, состоит из модулей, обеспечивающих унифицированный

доступ к описанию распределенной системы и трассе её работы, модуля управления визуализатором и модулей-преобразователей, каждый из которых осуществляет определенный вид отображения.

Временная диаграмма

Основными видом отображения является, так называемое, отображение временной диаграммы в виде "линий жизни" отдельных компонентов. Здесь компоненты системы отображаются в виде именованных горизонтальных линий, на которых отображаются события, состояния и взаимодействия соответствующих компонентов.



Под линиями жизни компонентов расположена временная шкала, которая, благодаря наличию вертикальной визирной линии, позволяет легко определить момент времени работы системы, соответствующий заданной точке на линии жизни. Слева от линий жизни расположены имена соответствующих им компонентов распределенной системы. Строка статуса отображает значение времени работы системы, соответствующее положению курсора, а также текущее состояние текущего компонента.

В процессе функционирования распределенной программной системы в различных ее компонентах происходят события. Для исследователя важно иметь возможность визуально оценивать распределение событий по компонентам и по времени, учитывая при этом и другие параметры событий. События, произошедшие в компонентах системы, отображаются на соответствующих линиях жизни. Каждое событие изображается в виде пересекающей линию жизни вертикальной черты, над которой расположена буква, указывающая на тип данного события.

Разные типы событий могут нести различную дополнительную семантическую нагрузку, например, события начала и завершения состояний и события взаимодействия компонентов.

Взаимодействия компонентов отражаются в трассе работы системы посредством комбинаций событий определенный типов. Соответственно, при визуализации эти взаимодействия воссоздаются из разрозненных событий в разных компонентах.

Визуализатор позволяет отображать как простые связи «один к одному» (в виде стрелки) или «один ко многим» (в виде грозди стрелок), так и более специфические виды взаимодействий, например, барьеры.

Барьеры - это события синхронизации нескольких компонентов. Каждый из этих компонентов входит в состояние ожидания барьера, а когда все они находятся в этом состоянии, все компоненты одновременно покидают барьер.

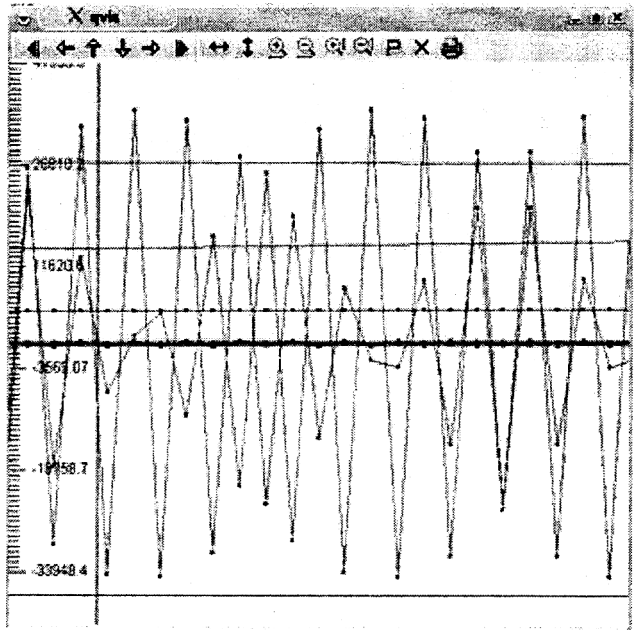
Состояние ожидания барьера отображается двумя красными параллельными горизонтальными линиями, идущими над и под линией жизни компонента. Момент выхода компонентов из барьера отмечается вертикальной красной чертой, пересекающей линии жизни компонентов, участвующих в барьере.

В процессе функционирования распределенной программы ее отдельные компоненты могут переходить в различные состояния. Это отражается в трассе событий соответствующими событиями смены состояния. Каждому состоянию сопоставлен номер, название и цвет.

Состояние отображается окраской линии жизни компонента в соответствующий состоянию цвет и помещением поверх линии жизни его названия.

Отображение значений параметров системы

Помимо событий, состояний и взаимодействий компонентов визуализируемой системы исследователю бывает необходимо отслеживать динамику изменения значений некоторых переменных и/или параметров системы. Визуализацией этой информации занимается дополнительный вид отображения, строящий графики изменения данных значений.



Для удобства исследования значений переменных действительного типа визуализируемой системы предусмотрен режим интерполяции значений между соседними точками, что бывает полезным, так как значения переменных попадают в трассу дискретно, через некоторые промежутки времени.

В режиме отображения графиков переменных появляется дополнительная вертикальная шкала значений переменных и дополнительный горизонтальный визир.

Механизмы детализации исследуемой области и навигации по ней

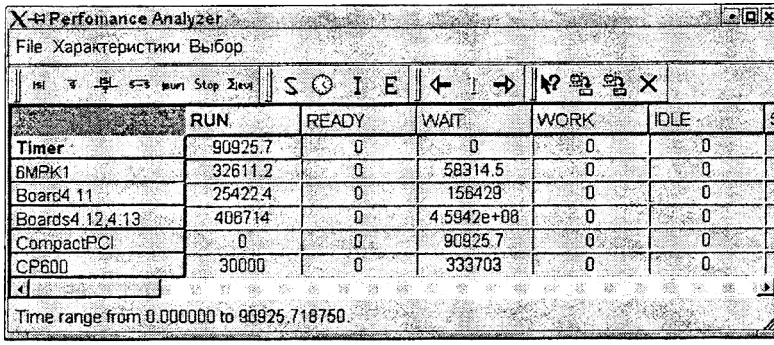
Визуализатор предоставляет гибкие механизмы навигации по исследуемой области и навигации по ней. Это:

- удобные возможности навигации по времени работы системы.
- механизмы выбора отображаемых типов событий, состояний, сообщений и компонентов.
- механизм семантического слияния компонентов [4], позволяющий удобно группировать компоненты, то есть, объединять линии жизни различных компонентов в одну с тем, чтобы, например, анализировать взаимодействия между кластерами компонентов или оценивать функционирование интерфейсных компонентов.
- механизм поиска событий и состояний по заданной маске.
- механизм интеграции с исходным кодом программы, позволяющий для любой точки времени и любого компонента посмотреть список близких событий и для каждого из них показать участок исходного кода, его породивший.

4.2. Анализатор производительности распределенной системы

Анализатор производительности распределенной системы является инструментом, позволяющем собирать статистическую информацию по трассе работы распределенной программной системы. Такая информация бывает необходима как для проверки корректности работы системы, так и для изучения её свойств, например, загруженности отдельных компонентов или относительной продолжительности различных состояний.

Результирующие данные представляются в виде таблицы и могут быть экспортированы во внешний файл в формате csv.



Характеристики производительности

- Анализатор производительности предназначен для вычисления ряда типовых характеристик производительности, выявленных и сформулированных при работе над несколькими разноплановыми практическими проектами.
- Среди таких характеристик:

- Общая продолжительность состояния (вычисляемая, как суммарная продолжительность заданного состояния данного компонента на исследуемом промежутке времени).
- Максимальная продолжительность интервала неизменности данного состояния данного компонента на заданном отрезке времени.
- Суммарная продолжительность заданного состояния данного компонента на исследуемом промежутке времени, деленная на длину этого промежутка.
- Максимальный интервал между соседними моментами начала состояния одного типа на заданном промежутке времени.
- Суммарное число событий каждого типа, произошедших в данном компоненте на заданном промежутке времени.
- Максимальное и минимальное значения данного параметра системы на заданном промежутке времени.
- Матожидание и дисперсия значения данного параметра системы на заданном промежутке времени.

Анализатор производительности предоставляет гибкие механизмы детализации исследуемой области и навигации по ней, аналогичные механизмам, реализованным в визуализаторе.

4.3. Инструменты для работы с трассой

Помимо визуализатора и анализатора производительности в состав среды визуализации входят инструменты для предварительной обработки трасс работы системы. Среди них:

- Средство слияние нескольких трасс в одну – применяется для получения общей трассы распределенной системы из трасс собранных локально на исполнителях.
- Инструменты для индексации трассы, с помощью которых осуществляется предварительная обработка трассы перед ее визуализацией и анализом производительности
- Отладочные утилиты, используемые для просмотра и поиска по трассе в текстовом виде.

5. Практическое применение системы

Построенные средства были апробированы в ряде практических проектов:

- в проекте "WebES", проведенном совместно с Эйнтховенским университетом (Нидерланды), данный инструментарий применялся для проверки соответствия построенной модели специфицированным требованиям (предоставленным в виде диаграмм MSC).
- при разработке средств моделирования аппаратных среств и программного обеспечения для специализированного вычислителя данные средства вошли в пакет инструментов, поставляемых заказчику.
- в создаваемой среде разработки моделей для стенда полунатурого моделирования данные средства вошли в пакет инструментов, поставляемых заказчику.

Заключение

Построенная среда визуализации в данный момент успешно применяется в нескольких различных практических проектах, что позволяет говорить о её востребованности и достаточной настраиваемости. В настоящий момент рассматривается возможность применения данной среды для визуализации истории событий, собираемых на локальных датчиках системы автоматического обнаружения вторжений [5].

Перспективы развития среды визуализации включают в себя:

- реализацию алгоритма семантической склейки компонентов в качестве стороннего инструмента,
- разработку и реализацию алгоритмов сравнения трасс,
- интеграцию анализатора производительности с визуализатором для построения единого удобного инструмента изучения трассы,

Литература

1. Тихонов А.В. Разработка среды визуализации функционирования распределенных программ. // Дипломная работа, МГУ ВМиК, 2002. http://zigzag.lvk.cs.msu.su/~tikhonov/diplom.doc
2. B.Price,R.Baecker,I.Small,"Principled Taxonomy of Software Visualization"(c)1994
3. R.J.Hendley, N.S.Drew, A.M.Wood & R.Beale, "Narcissus: Visualising Information" (c)2000;
4. Тихонов А.В. Семантическое сжатие информации при визуализации функционирования распределенных вычислительных систем // Труды Всероссийской научной конференции "Методы и средства обработки информации" (октябрь 2003 г., г. Москва) -М.: Издательский отдел факультета ВМиК МГУ, 2003. - С. 503-508.
5. Гамаюнов Д.Ю. Современные некоммерческие системы обнаружения атак // "Программные системы и инструменты": Тематический сборник факультета ВмиК МГУ им.Ломоносова N3, Под редакцией Л.Н.Королева –М: Издательский отдел факультета ВмиК МГУ 2002.

Раздел III
Параллельные и распределенные вычисления

Машечкин И.В., Попов И.С., Смирнов А.А.

Автоматизированная библиотека контрольных точек для кластерных вычислительных систем[3]

1. Введение

При использовании кластерных вычислительных систем (где моделью параллельного программирования является модель обмена сообщениями [2]) возникает задача эффективного использования её ресурсов. Эту задачу помогает решить механизм контрольных точек. *Контрольная точка* – информация о состоянии задачи, которая позволяет продолжить ее выполнение с некоторого момента времени. Основной целью применения механизма контрольных точек является повышение эффективности использования кластерного времени. Такое повышение эффективности может быть достигнуто путем повышения отказоустойчивости вычислительных задач [1], а также путем обеспечения большей гибкости планирования для системы управления заданиями. В первом случае периодически создаваемые контрольные точки задачи позволяют свести к минимуму потери при сбое в кластере. Во втором случае механизм контрольных точек предоставляет системе управления заданиями возможность временно снимать со счета длительные вычислительные задачи, освобождая ресурсы для задач с более коротким временем счёта, а затем путем восстановления из контрольной точки продолжать выполнение длительных задач (при этом возможна ситуация, при которой задача будет восстановлена и продолжит работу другом наборе узлов по сравнению с тем набором, на котором она была запущена изначально). Также в ситуации, когда длительные задачи прерываются на время в пользу более коротких, может улучшаться качество обслуживания пользователей вычислительной системы, если под улучшением качества обслуживания понимать минимизацию времени, которое задача проводит в очереди готовых к запуску при незначительном увеличении времени счета других задач.

При применении механизма контрольных точек для параллельных задач в системе обмена сообщениями возникают особые

[3] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 02-01-00261

проблемы, не свойственные системам контрольных точек для последовательных задач. Дело в том, что возможны ситуации, при которых даже корректное восстановление состояния всех ветвей задачи приведет к некорректному продолжению вычислений, что может быть связано с потерей или дублированием сообщений в коммуникационной среде. Кроме проблемы обеспечения корректности восстановления для параллельных задач актуальной является задача повышения эффективности самого механизма контрольных точек. Под повышением эффективности системы контрольных точек понимается уменьшение издержек на создание контрольной точки и восстановление из неё. Очевидно, что контрольные точки должны быть реализованы таким образом, чтобы их применение не оказывало существенного влияния на время выполнения задачи.

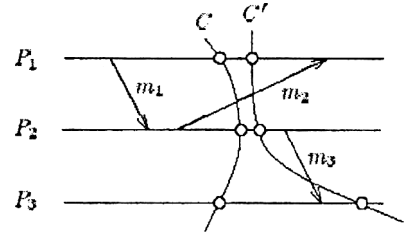
2. Подходы к реализации механизма КТ

Все механизмы контрольных точек по способу реализации можно разделить на два класса: *пользовательские* контрольные точки и *автоматические* контрольные точки. В случае применения первого подхода программист для каждой конкретной задачи реализует механизм контрольных точек «с нуля», полностью решая задачу сохранения и корректного восстановления состояния задачи. Автоматические контрольные точки реализуются на уровне операционной системы или на уровне библиотеки программирования, такая реализация предлагает услуги по созданию контрольных точек для широкого класса задач, требуя от программиста внесения минимальных изменений в исходный код задачи. Реализация на уровне операционной системы (в виде модуля ядра ОС) позволяет более полно сохранять и восстанавливать состояния процесса, однако такая реализация предполагает относительно низкую переносимость (система контрольных точек обращается к структурам данных ядра ОС, которые подвержены частым изменениям при выходе новых версий ОС) и требует прав администратора при внедрении; реализация же в виде библиотеки, которая работает на уровне процесса пользователя, использует менее подверженные изменениям интерфейсы и не требует участия администратора в процессе внедрения, однако позволяет сохранить лишь подмножество состояния процесса, которого, однако, оказывается достаточно для большинства вычислительных задач. Если же сравнивать пользовательские и автоматические контрольные точки, то можно сказать, что пользовательские контрольные точки обычно являются более эффективным механизмом, так как их реализация может учитывать специфические особенности конкретной задачи, однако необходимость реализации для каждой задачи является серьезным недостатком. Автоматические контрольные точки в общем случае могут

предложить более низкую эффективность, но это может компенсироваться общностью подхода.

Как уже упоминалось ранее, чаще всего параллельные задачи в кластерных вычислительных системах взаимодействуют при помощи обмена сообщениями через общую коммуникационную среду, которой в кластерах является сеть, либо специализированная (такая, как Myrinet, SCI), либо обычная (например, Fast Ethernet). Это означает, что в состояние параллельной задачи (а значит и в контрольную точку этой задачи) входит кроме состояния каждой её ветви как отдельного процесса еще и состояние коммуникационной среды. Обычно состояние коммуникационной среды (сообщения, которые могли в ней находиться в момент создания контрольной точки) не сохраняется, так как его сохранение и восстановление потребует значительных усилий, вместо этого вводятся ограничения на это состояние.

Рассмотрим следующий пример:



На данном рисунке показаны три ветви параллельно задачи P, которые взаимодействуют при помощи отправки сообщений m. Горизонтальные линии соответствуют времени в каждой ветви, кружочками отображены моменты создания контрольных точек. Так, при восстановлении из контрольной точки C, будет потеряно сообщение m_2, а при восстановлении из контрольной точки C' будет потеряно m_2 и продублировано сообщение m_3. Таким образом, при восстановлении возможны два типа ошибок – потеря и дублирование сообщений, обе эти ошибки для вычислительных задач приведут к некорректному продолжению вычислений (в других применениях, в случае взаимодействующих при помощи сообщений приложений цена потери и дублирования может быть другой). Поэтому будем называть в дальнейшем контрольную точку вычислительной задачи *консистентной*, если в момент её создания не было обменов сообщениями, то есть все отправленные сообщения были получены, и в каналах связи не было сообщений. Во всех других случаях контрольная точка будет называться *неконсистентной*.

Существует два классических алгоритма обеспечения консистентности: *синхронные* и *асинхронные* контрольные точки. В первом случае происходит инициализация процедуры создания

контрольной точки в одной из ветвей, которая отправляет служебное сообщение всем остальным ветвям: «Готовьтесь к созданию контрольной точки». После получения такого сообщения ветви прекращают отправлять сообщения, за исключением служебных, но продолжают принимать сообщения. Затем, когда все ветви убеждаются в том, что коммуникационные каналы пусты (например, при помощи тестовых сообщений), они сообщают о своей готовности ветви-инициатору. Как только все ветви готовы к созданию контрольной точки, они одновременно начинают создавать свои контрольные точки. После того, как создание контрольных точек завершено, выполнение параллельной задачи продолжается.

Асинхронные контрольные точки создаются в ветвях задачи в произвольные моменты времени, причем в каждой ветви независимо. При этом пропадает всякая необходимость в синхронизации процессов, однако получение консистентной контрольной точки из набора контрольных точек ветвей не всегда является тривиальной задачей. При таком подходе требуется хранить несколько (а иногда и все) контрольные точки для каждой ветви. Поиск консистентного состояния на наборах контрольных точек является задачей отдельного алгоритма, но при его использовании возможны ситуации, когда ни одна комбинация контрольных точек отдельных ветвей не образует *консистентного* состояния (*эффект домино*).

Рассмотрим еще один подход: *предварительная синхронизация*, гарантирующая консистентность контрольной точки. В этом случае в исходном коде задачи заранее размещаются подсказки для системы контрольных точек, которые говорят о том, в какой именно момент ветви параллельной задачи находятся в таком состоянии, что их контрольные точки будут образовывать консистентное состояние. Выбор места синхронизации основывается на следующих критериях:

- коммуникационные каналы пусты;
- никакой процесс не заблокирован в состоянии получения сообщения.

Эти условия, очевидно, гарантируют консистентность контрольной точки, так как ни одно сообщение не будет потеряно (первое условие) и все процессы готовы к созданию контрольной точки (второе условие). Такие точки синхронизации обычно располагаются в конце итерации вычислений, после выполнения групповых операций MPI, которые затрагивают все ветви задачи. Данный подход за счет предварительного выбора предполагает минимальные временные затраты на синхронизацию ветвей задачи перед созданием контрольной точки.

Для осуществления выбора места синхронизации может быть предложено несколько методов:

- ручной выбор программистом;

- статический анализ программы (места синхронизации выявляются на этапе компиляции);
- динамический анализ программы (выполняется несколько тестовых прогонов программы, во время которых обнаруживаются самые удачные точки синхронизации).

Под повышением *эффективности* системы контрольных точек понимают уменьшение издержек на создание контрольной точки и восстановление из неё. Основная часть издержек при создании контрольной точки связана с записью данных на диск, а объем записываемых данных можно оценить объемом памяти, используемым параллельной задачей (суммарным объемом данных всех её ветвей). Издержки на восстановление из контрольной точки связаны с чтением того же объема данных, дополнительными процедурами при восстановлении, но всё же большая часть издержек связана с вводом-выводом. Следовательно, повышение эффективности можно ожидать при уменьшении издержек на чтение и запись данных на диск (или на файловый сервер, в зависимости от конфигурации кластера), а также при уменьшении самого объема данных, который необходимо записать. Рассмотрим несколько методов повышения эффективности механизма контрольных точек; описание других методов можно найти, например, здесь: [3,4].

Сжатие данных. Так как контрольная точка содержит данные процесса, а они достаточно хорошо сжимаются (в предположении об избыточности данных), то применение алгоритма сжатия к содержимому контрольной точки позволяет значительно уменьшить её размер. Однако так как цель состоит также в уменьшении задержек на время создания контрольной точки, необходимо выбирать алгоритмы, которые бы обеспечивали приемлемые коэффициенты сжатия, но при этом не требовали большого количества процессорного времени. Такой компромисс можно найти в семействе алгоритмов словарного сжатия. Если заранее известен характер данных задачи, можно применять специализированные алгоритмы сжатия, которые бы позволили достичь более качественных результатов на конкретных задачах.

Обычный процесс создания контрольной точки имеет существенный недостаток: выполнение процесса прерывается на время создания контрольной точки. В случае *асинхронной* контрольной точки выполнение задачи не прерывается, а создание контрольной точки ведётся в фоновом режиме, однако, так как требуется записать состояние процесса на момент создания контрольной точки, необходимо каким-то образом сохранить это состояние. В ОС семейства UNIX сохранение состояния может быть достигнуто экономным способом, при помощи системного вызова fork(). Так как создание контрольной точки – это в основном ввод-вывод, а самой

вычислительной задаче требуется время ЦП, то эти операции можно выполнять параллельно без потери производительности.

Еще одним способом повышения эффективности является *исключение ненужных областей памяти из контрольной точки.* Этот способ является актуальным, если после создания контрольной точки задача первой операцией обращается к некоторым достаточно большим областям памяти на запись, а не на чтение. В этой ситуации нет необходимости включать эти области памяти в контрольную точку, так как после восстановления сохраненные значения не будут использованы. Примером таких областей памяти могут служить временные массивы данных, используемые задачей на каждой итерации вычислений, при условии, что контрольная точка создается в конце итерации.

3. Экспериментальная реализация: libcheckpoint

В результате анализа существующих решений, а также исследования различных подходов к реализации систем контрольных точек был выбран следующий подход: библиотека автоматических контрольных точек на уровне процесса пользователя ОС семейства UNIX для реализации MPI — MPICH. Был осуществлен выбор метода обеспечения консистентности контрольной точки: предварительная синхронизация – данный метод позволяет обеспечить низкие задержки на синхронизацию ветвей, требует хранения лишь одного набора контрольных точек ветвей и его реализация не приводит к изменению структуры обмена сообщений задачи (не требует применения служебных сообщений). В библиотеке были реализованы три наиболее применимых метода повышения эффективности: сжатие данных, асинхронные контрольные точки и исключение ненужных областей данных из контрольной точки.

При проектировании и разработке преследовалось несколько основных целей:

- переносимость;
- отсутствие необходимости внесения изменений в системные библиотеки и библиотеки MPICH (или необходимость минимальных изменений);
- настройка на конечную архитектуру на этапе компиляции;
- максимальная эффективность при сохранении корректности сохранения состояния задачи.

Основными характеристикам библиотеки являются:

- Реализация автоматических контрольных точек на уровне процесса пользователя; сама библиотека libcheckpoint написана на языке C.

- Поддержка различных архитектур: FreeBSD/x86, Linux/x86, Linux/Alpha.
- Поддержка языков программирования C/C++, Fortran.
- Сохранение состояния процесса (сегмент данных, стек, куча, контекст процесса).
- Сохранение состояния операционной системы (отображения в память (`mmap/munmap`), открытые файлы (`open/close/lseek/read/write/...`), обработчики сигналов).
- Миграция процессов между машинами с идентичной архитектурой (восстановление задачи на другом наборе узлов по сравнению с тем, на котором она работала до создания контрольной точки).
- Получение информации об эффективности работы библиотеки.

3.1. Архитектура библиотеки

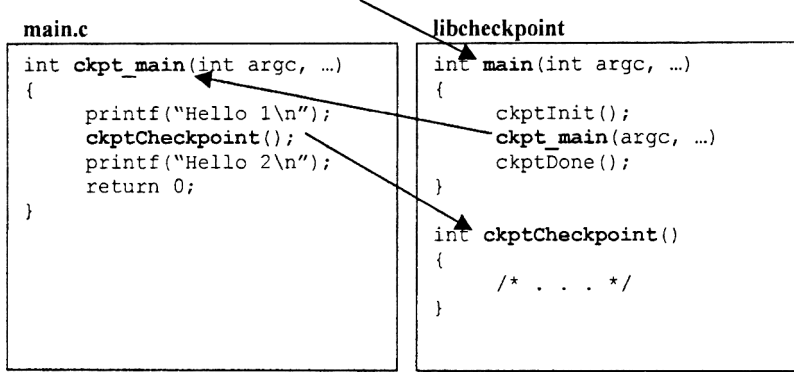
Архитектуру библиотеки можно разбить на следующие уровни:

- *Подсистема чтения/записи файлов контрольных точек.* Данный слой отвечает за формирование файла контрольный точек, который состоит из заголовка и информации о каждой сохраненной области памяти (её тип, размер, начальный адрес и содержимое). Эта подсистема также отвечает за нумерацию файлов контрольных точек.
- *Подсистема сжатия контрольных точек* отвечает за сжатие содержимого сохраняемых областей памяти. Сжатие используется для уменьшения размера контрольной точки. Данная подсистема может быть полностью отключена при настройке библиотеки.
- Подсистема *поддержки списка областей адресного пространства* обеспечивает изменение, сохранение и восстановление списка областей адресного пространства, которые необходимо включить или исключить из контрольной точки. Данный список может модифицироваться как другими подсистемами библиотеки, так и пользователем через специальный интерфейс (чтобы обеспечить возможность программисту исключать ненужные области памяти из контрольной точки).
- Подсистема *определения расположения различных областей процесса* инициализирует и поддерживает список областей адресного пространства процесса, внося туда информацию о расположении сегмента данных, стека и кучи процесса.

- Подсистема *перехвата обращений к библиотечным вызовам* позволяет библиотеке запоминать информацию обо всех системных ресурсах, используемых процессом. Информация о ресурсах заносится в специальные таблицы и используется при восстановлении, также может модифицироваться список областей адресного пространства процесса (например, при выделении и освобождении отображений в память).
- Подсистема *интерфейса с библиотекой MPICH* обеспечивает синхронизацию, корректное сохранение и восстановление состояния библиотеки обмена сообщениями при создании и восстановлении из контрольной точки. (Эта подсистема может быть отключена, тогда получается библиотека КТ для отдельных процессов).
- Подсистема *сбора информации об эффективности библиотеки* сохраняет информацию о продолжительности работы различных этапов работы библиотеки (создание контрольной точки, восстановление из контрольной точки, синхронизация).

3.2. Передача управления при работе библиотеки

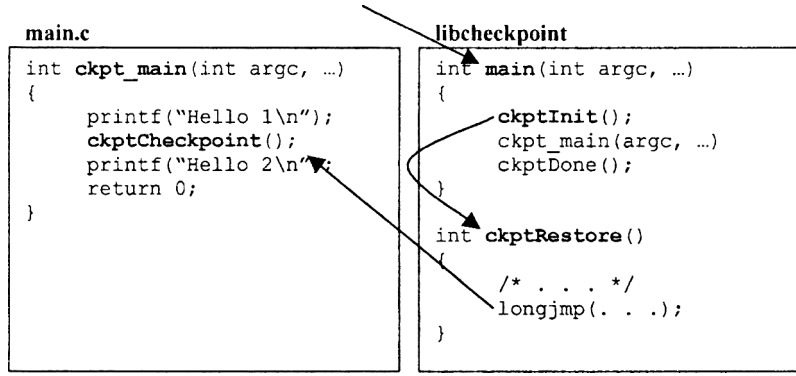
При запуске программы без восстановления из контрольной точки схема передачи управления между программой и библиотекой контрольных точек выглядит следующим образом:



После запуска программы управление передается в функцию `main()`, которая находится в библиотеке `libcheckpoint`. В случае если использование библиотеки не было запрещено при помощи конфигурационного файла, вызывается функция `ckptInit()`, которая кроме всего прочего проверяет наличие контрольной точки программы. Если контрольная точка не была найдена, управление передаётся в функцию `ckpt_main()`, которая уже находится в основной программе. После этого основная программа в какой-то момент может

вызывать функцию создания контрольной точки, `ckptCheckpoint()`, которая сохранит контрольную точку и вернет управление обратно в программу.

При восстановлении последовательность вызовов изменяется:



Функция `ckptInit()` обнаруживает наличие контрольной точки и вызывает функцию `ckptRestore()`, которая в конце восстановления вызывает библиотечную функцию `longjmp()`, которая уже передает управление основной программе в точку, соответствующую возврату из функции `ckptCheckpoint()`, после чего выполнение продолжается. (Данная схема несколько упрощена по сравнению с реальной последовательностью вызовов.)

4. Результаты тестирования эффективности

Для тестирования эффективности библиотеки была выбрана вычислительная задача расчета свободномолекулярного обтекания цилиндра, написанная на языке Fortran-77. Тестирование производилось на супер-ЭВМ МВС-1000М, программа запускалась в штатном режиме работы суперкомпьютера, поэтому на приведенные результаты существенное влияние могла оказать загруженность файлового сервера, связанная с работой других задач. Библиотека контрольных точек была сконфигурирована с использованием подсистемы сжатия. Измерения проводились при помощи встроенных в библиотеку средств, в каждом эксперименте задача создавала 10 контрольных точек на протяжении своей работы, после этого временные затраты усреднялись. В этом тестировании не использовались никакие методы повышения эффективности, за исключением сжатия данных контрольных точки.

Были получены следующие результаты:

К ол-во ветвей:	Р азмер КТ одной ветви, Кб:	Об щий размер КТ задачи, Кб:	Затраты (с) на:		того (с)
			со здание КТ	с инхронизац ию	
4	5 2063	2082 52	1 0,84	0, 51	1,34
10	2 3292	2329 20	7, 68	0, 67	,35
20	1 3854	2770 80	8, 93	0, 55	,48
30	9 570	2871 00	9, 11	0, 51	,62
40	9 234	3693 60	7, 35	0, 60	,95
В среднем:			**8, 78**	**0, 56**	**,35**

Из полученных результатов следует, что в среднем издержки на создание контрольной точки такой задачи составляют 9,35 секунды, что при интервале создания контрольных точек в 30 минут составляет 0,5% потери по времени. Коэффициент сжатия данных контрольной точки в данном эксперименте достигал 43%.

Среднее время восстановления задачи составляет около 15 секунд, что является вполне приемлемым.

5. Заключение

На основе проведенных исследований была выполнена экспериментальная реализация библиотеки автоматических контрольных точек для параллельных задач в системе обмена сообщениями. Предлагаемая реализация поддерживает библиотеку обмена сообщениями MPICH в трех коммуникационных средах:

- разделяемая память (SMP-узлы);
- сеть Myrinet (используется в кластерных системах, например, МВС-1000М);
- сеть Ethernet.

Одной из целей при разработке являлось обеспечение переносимости полученной библиотеки: так, на настоящий момент поддерживаются следующие архитектуры вычислительных систем: FreeBSD/x86, Linux/x86, Linux/Alpha. К библиотеке разработан интерфейс для языков C/C++ и Fortran, то есть она может быть использована в вычислительных задачах, написанных на этих языках. В перспективе могут быть разработаны интерфейсы для других языков программирования.

Библиотека полностью сохраняет состояние процесса, из состояния операционной системы сохраняются только открытые файлы и отображения в память. Такого сохраненного состояния хватает для

корректного восстановления выполнения большинства вычислительных задач.

Предложенная реализация не является единственной, существуют также другие реализации систем контрольных точек (например, [3,6,7,8]), однако они обладают рядом существенных недостатков таких, как отсутствие поддержки параллельных задач, малая переносимость и отсутствие поддержки MPICH.

Отметим возможные направления развития библиотеки:

1. Перенос библиотеки libcheckpoint на другие архитектуры вычислительных систем, в том числе и на другие коммуникационные среды. Возможна адаптация к другим реализациям MPI (например, LAM/MPI).
2. Автоматический выбор места синхронизации, гарантирующего консистентность контрольной точки.
3. Автоматическое исключение ненужных областей памяти из контрольной точки [5].

Литература

1. *Christine Morin, Isabelle Puaut.* A Survey of Recoverable Distributed Shared Memory Systems. - IEEE Trans. on Parallel and Distributed Systems. – Volume 8. – Issue 9. – pp. 959-969. – 1997.
2. *Message Passing Interface Forum.* MPI: A Message-Passing Interface Standard. Version 1.1, June 1995.
3. *James S. Plank, Micah Beck, Gerry Kingsley, Kai Li.* Libckpt: Transparent Checkpointing under Unix. In *Proceedings of the 1995 Winter USENIX Technical Conference.* – UT-CS-94-242.
4. *James S. Plank, Jian Xu, Robert H. B. Netzer.* Compressed Differences: An Algorithm for Fast Incremental Checkpointing. Technical Report CS-95-302, University of Tennessee, August 1995.
5. *James S. Plank, Micah Beck, Gerry Kingsley.* Compiler-Assisted Memory Exclusion for Fast Checkpointing. - IEEE Technical Committee on Operating Systems and Application Environments, Special Issue on Fault-Tolerance. – Winter 1995.
6. *Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, Eric Roman.* Parallel Checkpoint/Restart for MPI Applications.
7. *J. Duell, P.Hargrove and E.Roman.* The Design and Implementation of Berkley Lab's Linux Checkpoint/Restart, 2002.
8. *В.А. Абрамова, А.В. Баранов, М.Ю. Храмцов.* Библиотека интерфейсных вызовов (API) для организации контрольных точек (КТ) (Документация по ПО МВС-1000М).

Воронов В.Ю., Джосан О.В., Медведев М.А., Попова Н.Н.

Опыт внедрения современного математического программного обеспечения на платформе IBM Regatta

Введение

Существует класс крупных прикладных задач, требующих выполнения множества вычислений, таких как умножение матриц, решение систем линейных уравнений, поиск собственных значений и т.д. Одним из направлений для эффективного решения возникающих проблем является использование многопроцессорных вычислительных комплексов. Но в связи с этим возникла необходимость разработки специальных средств для упрощения написания параллельных программ. Изначально они были низкоуровневыми и требовали от математиков знания многих особенностей распределенного программирования, что было весьма неудобно и занимало много времени, хотя давало существенно более эффективный код.

На текущий момент существует множество библиотек различной функциональности, каждая библиотека обладает своими характерными особенностями. ATLAS, Aztec, BlockSolve95, Distributed Parallelization at CWP, DOUG, GALOPPS, JOSTLE, NAMD, P-Sparslib, PIM, ParMETIS, PARPACK, PBLAS, PETSc, PGAPack, PLAPACK, ScaLAPACK, SPRNG – это далеко не полный список разработок. Постоянно появляются новые пакеты, каждый из которых позволяет преодолеть еще один шаг в направлении автоматизированного создания параллельных программ.

Представляет интерес задача исследования различных библиотек на предмет скорости математических вычислений на данной платформе и времени разработки программ.

Сравнение библиотек проводится на примере конкретной классической задачи (точную формулировку приводим ниже). Стоит отметить наличие зарубежных исследований в этой области [1].

Цель нашего исследования – провести сравнительный анализ некоторых достаточно новых разработок в этой области; показать, что новые библиотеки улучшают показатели и по скорости счета, и по удобству разработки приложения. Также интересно осветить проблемы, с которыми сталкивается пользователь при работе с библиотеками.

Постановка задачи

В качестве тестовой задачи в работе рассматривается трехмерная задача Пуассона:

$$\frac{\partial^2 U}{\partial x^2}+\frac{\partial^2 U}{\partial y^2}+\frac{\partial^2 U}{\partial z^2}=-f(x,y,x)$$

$$U_{\Gamma}=0$$

в области $\Omega=[0,1]\times[0,1]\times[0,1]$

Область Ω размельчается трехмерной сеткой 100x100x100 и для уравнения формулируется разностная схема. Таким образом, используя 7-точечный шаблон для численного решения исходной задачи, получаем разреженную матрицу размером 10^6x10^6 с числом ненулевых элементов порядка $7*10^6$.

Приведем схематическое изображение структуры исследуемой матрицы:

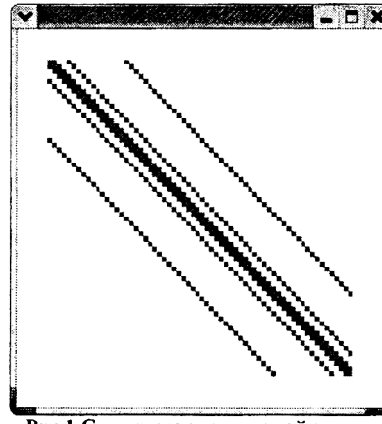


Рис.1 Структура исследуемой задачи

В качестве правой части системы уравнений выбирался вектор, отвечающий заранее известному решению уравнения Пуассона.

Выбор данной задачи в качестве тестовой обусловлен её широкой областью применения на практике, также это в своем роде классическая задача для применения численных методов. В то же время ясно, что тестирование на задаче такого вида не может объективно охарактеризовать качество реализации методов библиотек.

Обзор методов

Прежде чем более подробно рассказать об особенностях каждой из библиотек, отвлечемся немного на математический аппарат и приведем краткий обзор реализованных методов.

Наиболее эффективными и устойчивыми среди итерационных методов решения больших разреженных систем линейных

алгебраических уравнений с матрицами нерегулярной структуры являются так называемые проекционные методы, и особенно тот их класс, который связан с проектированием на подпространства Крылова[2,3]. Эти методы обладают целым рядом достоинств: они устойчивы, допускают эффективное распараллеливание, работу с различными строчными форматами и предобуславливателями разных типов.

В современных итерационных методах применяется связь предобуславливателя с решателем. Далее приведена общая схема.

Пусть решается СЛАУ Ax=f с невырожденной квадратной матрицей и ненулевой правой частью. Пусть M – некоторая невырожденная матрица размерности n. Тогда $M^{-1}Ax = M^{-1}f$ т.е. получается новая система: $\hat{A}x = \hat{f}$. Хотя данные системы эквивалентны, новая матрица в левой части имеет другие спектральные характеристики. Таким образом, целью обуславливателей является улучшение характеристик матрицы для увеличения скорости сходимости итерационного процесса. При этом матрица M должна удовлетворять следующим свойствам:

Должна быть по возможности "близка" к матрице A

Должна быть легко вычислима

Должна быть легко обратима

Данный вид предобуславливания называется левым. Другой подход связан с переходом к такой системе: $AM^{-1}y = b$. Подобная схема называется правой.

Типичным примером предобуславливателя является неполная LU факторизация (ILU(0)). При классической LU факторизации матрицы L и U заполнены более плотно, чем исходная матрица (это связано с самим процессом факторизации). Поэтому более выгодно осуществлять следующее разделение: A = L_aU_a+R, где в L_a и U_a заполнены те же элементы, что и в матрице A, а R – «остаток» от такого разложения. Таким образом, A заменяется на L_aU_a с некоторой погрешностью. Преимуществом является экономия памяти и удобство решения получившейся системы. Недостатком – то, что L_aU_a не точно заменяет матрицу A. Для того, чтобы «остаток» R становился меньше, используют так называемое многоуровневое ILU разложение (ILU(k)). В этом случае в L_a и U_a элементы заполняются в зависимости от уровня разложения. Результатом является большее потребление памяти, но разложение является более точным.

Типичными итерационными методами являются метод Якоби и Гаусса-Зейделя. К методам Крыловского типа относятся:

Ортогонализация Арнольди: $x_m = x_0 + \beta V H_m^{-1} e_1$ Метод сводится к тому, что на каждом шаге **m** минимизируется невязка: $\min_y \|b - A(x_0 + V_m y)\|_2^2$

Метод бисопряженных градиентов: $x_m = x_0 + \beta V_m T_m^{-1} e_1$

Метод сопряженных градиентов – тот же метод бисопряженных градиентов, но для симметричной матрицы.

Метод CGS для невязки $r_m = p_m(A) r_0$ использует вместо $p_m(A)$ полином $p_m^2(A)$

Метод BCGSTAB для невязки использует следующее соотношение: $r_m = p_m(A) q_m(A) r_0$, где q_m выбирается так, чтобы $p_m q_m$ не содержит нечетных степеней.

Обзор параллельных математических библиотек и обоснование их выбора

При выборе библиотек мы руководствовались несколькими критериями. Во-первых, были выбраны библиотеки с широкой функциональностью. Затем, эти системы являются сравнительно новыми разработками, претерпевающими постоянные изменения. Наконец, немаловажным фактором при выборе инструментов является простота использования и переносимости уже существующего последовательного кода, хотя этот фактор оценивается достаточно субъективно.

Таким образом, среди огромного многообразия библиотек [4] наш выбор остановился на инструментах PETSc и Hypre.

Hypre

Hypre (High Performance Preconditioners) – программная библиотека, предназначенная для решения линейных систем с разреженными матрицами большой размерности на системах с массовым параллелизмом. Hypre включает в себя несколько наиболее часто используемых итеративных методов на базе методов Крылова, которые можно использовать вместе с масштабируемыми предобуславливателями. Hypre поддерживает интерфейсы на C и Fortran. Для параллельного выполнения и использования нитей требуется поддержка MPI и OpenMP соответственно. В настоящее время только часть методов библиотеки реализована с помощью OpenMP. Более подробная информация на сайте[5].

PETSc

PETSc (Portable Extensible Toolkit for Scientific applications) – это мощный набор средств для численного решения дифференциальных

уравнений в частных производных и схожих проблем высокопроизводительных вычислений. PETSc состоит из нескольких библиотек. Каждая библиотека оперирует с определенным семейством объектов. Модули PETSc работают с: множествами индексов, векторами, матрицами (обычно разреженными), распределенными массивами, методами подпространств Крылова, предобуславливателями, нелинейными решателями, пошаговыми решателями для дифференциальных уравнений в частных производных по времени. Переносимый расширяемый инструментарий для научных вычислений (PETSc) успешно продемонстрировал, что использование современных парадигм программирования может облегчить разработку крупномасштабных научных приложений на языках Fortran, C, и C++. PETSc состоит из нескольких библиотек (подобных классам C++).Подробнее см. [6].

Целевая архитектура

В качестве платформы для исследования поведения библиотек выбрана IBM Regatta eServer pSeries 690™. Это 16-процессорная вычислительная система архитектуры SMP, на базе процессоров IBM POWER4 производительностью 1,3 ГГц. Приведем краткий набор характеристик платформы.

Каждое вычислительное ядро состоит из двух 64-разрядных процессоров с кэшами L1 объемом 64 Кбайт. На каждом кристалле размещен кэш L2 объемом 1,5 Мбайт с общим доступом. Емкость кэша третьего уровня – 32 Мбайта на микросхему. Объем оперативной памяти - 64 Гбайт. Скорость передачи по коммутирующей матрице для межсоединений достигает 30-35 Гбайт/с. Суперскалярные CPU с выдачей 8 инструкций за такт, выполнением команд не в порядке следования и с предсказанием переходов. Более подробная информация в [7].

Методика проведения тестирования.

Напомним, что тестирование проводилось на задаче с фиксированным числом точек расчетной сетки равным 100x100x100.

В качестве основного параметра для оценки эффективности работы рассматривалось время решения задачи. Для теста выбран вариант, не требующий больших накладных расходов на задание начальных данных и представление результатов решения. Замер времени проводился внутренними средствами библиотек, позволяющими получать время выполнения разных фаз работы приложения (инициализации, факторизации, решения и.т.п.), а также других характеристик производительности.

Программы, реализующие данную задачу, были реализованы на языках программирования C и Fortran.

Результаты тестирования.

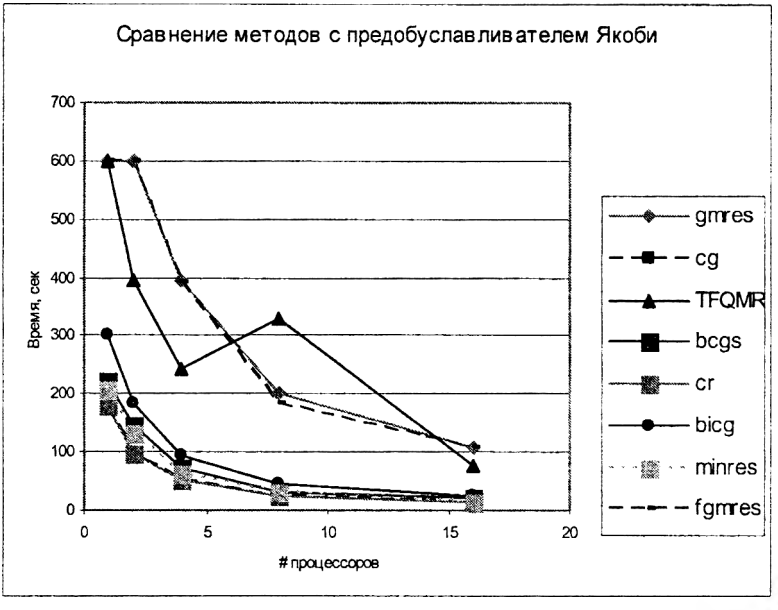
Далее мы приводим данные, полученные нами при тестировании. Представление результатов происходит по следующим показателям:

Продолжительность времени счета задачи для различных методов решения и количества процессоров.

Относительное ускорение работы метода при увеличении числа процессоров.

Результаты по библиотеке PETSc

Мы приводим результаты работы библиотеки PETSc на модельной задаче. Названия методов здесь и далее приводятся в сокращении, подробнее они описаны в [6].



Ри

с.2 Статистика времени работы методов PETSc при выбранном предобуславливателя Якоби.

Как видно из результатов, наиболее привлекательной по скорости работы является группа методов во главе с методом сопряженных градиентов(cg). Методы обобщенных невязок(gmres) и квазиминимальных невязок(tfqmr), напротив, показали себя не лучшим образом.

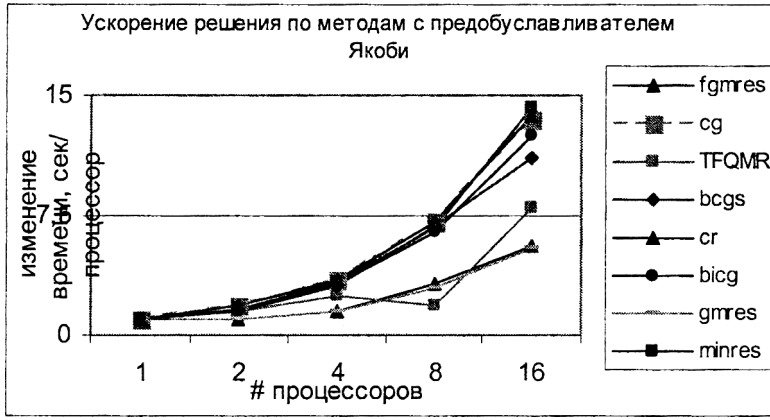


Рис.3. Ускорение работы методов PETSc при выбранном предобуславливателя Якоби.

По данным ускорения, наибольшую степень распараллеливаемости показывают несколько методов, среди которых самым лучшим оказался метод минимальных невязок (minres), а самым худшим - метод обобщенных невязок (gmres).

Результаты по Hypre

По аналогии с PETSc, приводим замеры времени для библиотеки Hypre. В этом случае представлена сводка методов вместе с предобуславливателями. Подробнее о методах и предобуславливателях в [5].

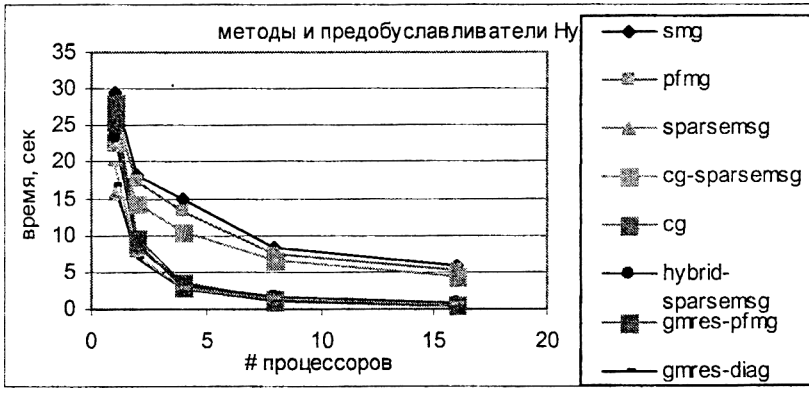


Рис.4. Время решения задачи для Hypre

По статистике работы стоит отметить с положительной стороны скорость работы разреженного метода SMG и тот факт, что время решения на Hypre примерно на порядок меньше, чем при использовании PETSc. Это можно объяснить особенностями SMP архитектуры платформы Regatta и тем, что в Hypre, в отличие от PETSc,

введена поддержка OpenMP, что заведомо дает Hypre существенное преимущество перед другими MPI-библиотеками.

Обратимся к картине ускорения для методов Hypre:

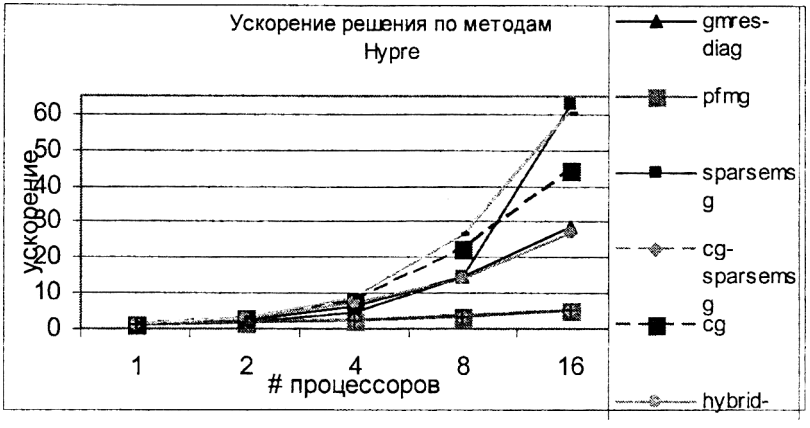


Рис.5. Ускорение работы методов Hypre

По ускорению наблюдаем похожую картину: хорошее ускорение разреженного SMG метода и очень слабые показатели группы методов smg, pfmg и cg-sparsemsg.

Сравнение и выводы

По результатам исследований была собрана таблица "лучших" методов библиотек (в нее попали методы, для которых время счета на 1 процессоре не превышает 150 секунд):

# проц / метод	1	2	4	8
H-smg	59.4	37.19	28.81	16.3
H-pfmg	53.41	35.51	26.82	15.71
H-sparsemsg	31.95	17.9	6.79	2.85
H-cg-sparsemsg	44.56	28.92	21.14	12.34
H-cg	56.6	18.9	6.1	2.95
H-hybrid-sparsemsg	46.52	16.79	7.15	2.21
H-gmres-pfmg	51.95	17.48	6.63	1.79
H-gmres-diag	33.26	12.47	5.16	1.92
H-gmres	39.52	15.28	5.49	3.97
P-	107.44	70.16	33.44	17.58

P-cg-Bjacobi				
P-cg-ASM	125.14	88.46	46.3	25.14
P-cr-Bjacobi	112.65	66.9	33.69	17.82
P-cr-ASM	131.12	83.74	45.27	25.94
P-minres-Bjacobi	117.81	80.34	38.26	20.23

Табл. 1. Сводное время решения задачи "лучшими" методами библиотек

Здесь префиксы P- и H- перед названиями методов обозначают библиотеки PETSc и Hypre соответственно.

Выводы

Наши исследования показали, что в библиотеке PETSc лишь небольшая часть методов подходит для эффективного решения СЛАУ с разреженными матрицами. Среди них рекомендуется использовать методы минимальных невязок, сопряженных градиентов и сопряженных невязок с блочным предобуславливателем Якоби. С лучшей стороны на данной задаче показала себя библиотека Hypre. Она заметно опережает PETSc в большинстве рассмотренных методов.

Наиболее эффективными для Hypre являются разреженный SMG метод и метод обобщенных невязок с выделением главной диагонали в качестве предобуславливателя.

При работе с библиотеками мы столкнулись с некоторыми проблемами.

Во-первых, это формат представления входных данных. Хотя существует несколько стандартных форматов для представления матриц (например, широко применяемый Compressed Sparse Row формат), каждая библиотека требует свое представление данных, и не в каждой есть возможность их преобразования из стандартного во внутреннее представление.

Вторая проблема – привязка к архитектуре. Некоторые библиотеки используют особенности аппаратного обеспечения, что не позволяет использовать их на платформах другой архитектуры. Например, мы сталкивались с определенными трудностями при переносе PETSc с платформы Regatta на МВС-1000М, которые, однако, были благополучно преодолены.

Авторы сердечно благодарят руководство факультета ВМиК МГУ за предоставленную возможность работать на платформе Regatta, администраторов Regatta за помощь во внедрении программных средств, участников лаборатории Интел-МГУ, а также члена-корреспондента РАН Л.Н. Королёва за ценные замечания и советы по ходу работ.

Работы проведены в рамках студенческой лаборатории Интел-МГУ при поддержке гранта РФФИ 02-07-90130.

Ссылки:

[1] - John Fettig, Wai-Yip Kwok, Faisal Saied "Scaling Behavior of Linear Solvers on Large Linux Clusters", 1999, Mathematics Department, Macquarie University, Sydney.

[2] - Самарский А. А. "Теория разностных схем", М., 1989.

[3] - «Матричные вычисления», Дж. Голуб, Ч. Ван Лоун, изд. «Мир», Москва, 1999

[4] - Netlib Repository, http://www.netlib.org/

[5]-HYPRE. High Performance Preconditioners. http://www.llnl.gov/CASC/hypre/

[6] - Portable Extensible Toolkit for Scientific Computation(PETSc), http://msc.anl.gov/petsc, http://www-unix.mcs.anl.gov/petsc/petsc-2/documentation/linearsolvertable.html

[7] «The POWER4 Processor Introduction and Tuning Guide», S. Behling, R. Bell, P. Farrell, H. Holthoff, F. O'Connell, W. Weir, IBM, Nov. 2001.

[8] - Message Parsing Interface, http://www.mpi-forum.org/

[9]- OpenMP. A Simple, Portable, Scalable SMP Programming, http://www.openmp.org/

[10] – IBM pSeries Regatta, http://www.regatta.cs.msu.su/p690.htm

Истомин Т. Е.

Обзор систем параллельного программирования на основе ограниченного набора типовых алгоритмических структур

Введение

Создание сложных систем обычно связано с выделением нескольких уровней абстракции. Абстрактные конструкции уменьшают свободу разработчика, но обеспечивают более удобное и подходящее для задачи представление ресурсов. На практике это разбиение на уровни приводит к потере производительности. С другой стороны, стоящая перед разработчиками проблема может быть настолько сложной, что ее решение с помощью низкоуровневых средств окажется практически невозможным.

Абстракция ограничена ожидаемым уровнем производительности системы. С ростом производительности аппаратного обеспечения это ограничение отодвигается и новые уровни абстракции получают право на жизнь, обеспечивая более удобное, а значит и быстрое создание все более сложных программных систем. В случае последовательного программирования новые уровни могут создаваться на основе старых, возможно "накопление опыта". В случае же параллельных вычислений, развитие аппаратуры может влиять на все уровни (последовательная программа только выиграет при переходе на более быстрый компьютер, а параллельная, может не ускориться или даже замедлиться при переходе от двух процессоров к тысяче).

Существует большое разнообразие подходов к построению высокоуровневых систем параллельного программирования. Начиная с полностью изолирующих программиста от параллелизма, и осуществляющих автоматическое распараллеливание (в основном это функциональные и логические языки). И заканчивая системами с абстракциями, жестко привязанными к конкретной архитектуре. Видны достоинства и недостатки этих подходов. В первом случае программист получает набор удобных абстракций, но полностью лишается возможности влиять на производительность алгоритма. Во втором, наоборот – предлагаются неудобные во многих случаях средства, но без потерь производительности. Между этими крайностями существует ряд промежуточных систем с различной степенью архитектурной зависимости.

Объединяет эти системы то, что предлагаемые ими модели программирования подразумеваются *универсальными*, в том смысле, что весь спектр средств может быть использован для описания любых вычислений. В более абстрактных системах это позволяет программисту описывать трудно распараллеливаемые вычисления, а в менее абстрактных – приводит к сложности применения конструкций.

Типовые алгоритмические структуры

Понятие *типовой алгоритмической структуры* (TAC, algorithmic skeleton) было введено Murray Cole в фундаментальной работе Algorithmic Skeletons: Structured Management of Parallel Computation[1]. Предложенный в данной работе подход основывается на отрицании универсальности модели программирования для избежания недостатков универсальных систем, но с сохранением высокого уровня абстракции и независимости от аппаратного уровня. Основная идея этого подхода заключается в том, что программисту принципиально разрешено создание только хорошо распараллеливаемых программ.

В качестве основной конструкции предлагается *функция высшего порядка* (higher order function). То есть функция F, параметром которой является другая функция f, и значением является функция g (композиция функции-параметра и каких-то других функций).

$$F : f \to g$$

Примером может служить функция *map*, применяющая функцию-параметр $f : a \to b$ ко всем элементам списка или массива $[a]$.

$$(map(f)) : [a] \to [b]$$

Подобные функции идеально вписываются в синтаксис функциональных языков. В императивных же языках они могут реализовываться как шаблоны (template) или функции от функций, если язык позволяет передачу функции в качестве параметра.

Работа программиста заключается в выборе подходящей для своей задачи функции высшего порядка (если таковая имеется в наличии), и реализации функций-параметров для нее. Только созданные таким

образом программы являются допустимыми. Предлагаемые в качестве базовых единиц программирования функции высшего порядка называются *типовыми алгоритмическими структурами (algorithmic skeletons).*

Очевидно, что применимость подобных систем ограничивается предлагаемым набором типовых алгоритмических структур.

С точки зрения пользователя системы (программиста) ТАС не выглядит параллельной (хотя не исключены и явно параллельные случаи), но за ней может скрываться высокоэффективная параллельная реализация. Аналогично тому, как для какого-либо языка программирования может существовать несколько компиляторов под различные аппаратные платформы, ТАС может иметь несколько функционально эквивалентных реализаций, оптимизированных под конкретную архитектуру вычислителя. С вариантом реализации типовой алгоритмической структуры может быть связана модель производительности, то есть функция от параметров вычислителя и характеристик функций-параметров. На основе этих моделей принимается решение об использовании одного из вариантов реализации ТАС -- лучшего для данной архитектуры вычислителя и конкретной задачи [4, 5, 6, 7]. Этим обеспечивается переносимость программ в смысле эффективности.

Примерами простых и широко распространенных ТАС являются: редукция, разделяй-и-властвуй, вычисление всех частичных сумм (scan), композиция функций (конвейер, pipe), пул процессов (*farm*)[2], вычисления на сетке, рекурсия[15], очередь задач[1].

Структурное параллельное программирование

Предложенная M. Cole модель является одноуровневой, то есть в ней не уделяется внимание функциям-параметрам и возможности их параллельной реализации. В последующих работах [2, 12, 14] было предложено развитие подхода, заключающееся в возможности иерархического построения программ из ТАС. То есть в качестве функции-параметра каждой ТАС может выступать экземпляр другой ТАС. Эта идея вполне естественна при ее рассмотрении на уровне функций высшего порядка, но реализация поддерживающих вложенность ТАС является непростым делом. Одна из причин – необходимость создания эффективной модели взаимодействия типовых алгоритмических структур. Причем этих моделей также может быть несколько, для поддержки различных архитектур вычислителя.

В работе S. Pelagatti [2] рассматривается так называемое *структурное параллельное программирование*. Основная идея состоит в том, что программисту предлагается базовый набор простейших ТАС и ограниченный набор средств для их комбинирования (в отличие от одноуровневого подхода, где количество ТАС может неограниченно увеличивается). Кроме того, программист лишается низкоуровневых операций **send/receive**, по аналогии с операцией **goto** в структурном последовательном программировании. Главная задача исследований – найти такой набор средств, чтобы система была максимально универсальной, при сохранении преимуществ программирования на ТАС.

Рассматриваемые системы

Существует много реализаций идеи программирования в терминах ограниченного набора типовых конструкций. Их наиболее полный перечень можно найти на веб-странице [16]. Для данного обзора были отобраны несколько наиболее непохожих друг на друга современных реализаций. Представленные системы отличаются и по используемой концепции и по масштабу – от небольших академических разработок (параллельные библиотеки или языки) до крупных коммерческих систем с развитым набором инструментальных средств.

p3l, SKIE и Skel-BSP

SKIE (Skeleton-based Integrated Environment) – коммерческая система, разрабатывается в рамках проекта PQE2000 – программно-аппаратной гетерогенной платформы, содержащей SIMD и MIMD вычислители.

Система состоит из визуального средства для создания программ из ТАС, компилятора, средств мониторинга и профилирования.

p3l (Pisa parallel programming language) – высокоуровневый структурный явно параллельный язык программирования, изначально предназначенный для массивно параллельных вычислений на компьютерах общего назначения класса MIMD с распределенной памятью, в дальнейшем расширенный для использования в гетерогенной среде SKIE (координационный язык SKIECL).

Параллельный алгоритм представляется как иерархическая композиция ограниченного набора конструкций языка, соответствующих различным формам параллелизма (типовым алгоритмическим структурам). Конструкции разбиты на три класса:

- параллельные по управлению: Farm(пул процессов), Pipe(конвейер);
- параллельные по данным: Map, Reduce, Scan, Comp(композиция);
- контрольные: Loop(итерация), Seq(последовательный).

Синтаксис языка p3l основан на синтаксисе C, и позволяет использование конструкций, типов и библиотечных вызовов языка C для реализации последовательных модулей (Seq). При описании модулей (то есть экземпляров ТАС), используется специальный синтаксис, позволяющий задавать типы входных и выходных параметров, вложенные структуры (стадии конвейера, функцию редукции и т.д.), способ декомпозиции параметров, степень параллелизма[3]. Все структуры работают с потоками данных, что обеспечивает возможность построения произвольной иерархической композиции. Листья иерархии (последовательные функции) задаются с помощью структуры Seq.

Система SKIE позволяет также использование параллельных листьевых модулей (Seq), причем написанных на различных языках (C, Fortran, Java; HPF или MPI для параллельных).

```
map mulmat in(int a[n][m], int b[m][k]) out(int c[n][k])
    inner_product in(a[*i][], b[][*j]) out(c[*i][*j])
end map
```

Листинг 1. Определение модуля типа Map для задачи перемножения матриц на языке p3l.

Модуль *Map* на листинге принимает на вход потоки матриц, разбивает их на части и рассылает по процессорам. Далее на каждом из процессоров запускается подструктура *inner_product*, результаты собираются в матрицу *c* и отправляются в выходной поток. Необходимость разбиения матрицы по какой-то из размерностей определяется символом * в соответствующей координате, последовательность [] означает передачу размерности целиком всем процессорам. Существует способ задания декомпозиции с перекрытиями.

В процессе компиляции производится статическая оптимизация дерева программы под конкретную архитектуру, которая заключается в применении правил эквивалентного преобразования дерева и выборе оптимальных вариантов реализации типовых алгоритмических структур, присутствующих в программе. Оптимизация осуществляется на основе параметров вычислительной системы и моделей

производительности, связанных с каждым из вариантов реализации ТАС.

Типовые алгоритмические структуры реализованы в виде шаблонов (implementation templates), описывающих структуру взаимодействия процессов. Компилятор на основе этих шаблонов строит граф процессов, соответствующий конкретному случаю (масштабу параллелизма) и отображает его на граф процессоров вычислителя. Далее графы подструктур объединяются в один большой, представляющий программу целиком[3].

Компилятор использует pragma-конструкции языка, позволяющие вручную задавать параметры распараллеливания и сообщать дополнительную информацию о программе: предполагаемые размеры динамических данных, предполагаемые скорости работы пользовательских (листьевых) модулей.

Примеры моделей производительности шаблонов могут быть найдены в работах [2, 8] и в работах [4, 5] для BSP-реализации. Модели производительности для BSP-платформы разработаны в рамках проекта **Skel-BSP**, также основанного на языке из семейства p3l. Использование параметров BSP-машины позволяет строить аналитические оценки для оптимального выбора масштаба параллелизма в зависимости от параметров BSP-машины. В работе [6] приводится схема оптимизации p3l-программ для BSP-машины.

Frame

Другой пример языка структурного параллельного программирования – координационный язык Frame[13]. Основная идея – использование высокоуровневых конструкций для спецификации общей структуры параллелизма, а именно правил, определяющих последовательность применения вычислительных функций на различных группах процессов.

Программа состоит из двух частей: в первой части специфицируется параллельная структура алгоритма, а именно динамика разделения и объединения групп процессов и вызова вычислительных функций в этих группах. Во второй части на базовом языке программирования определяются вычислительные функции, которые могут быть как последовательными, так и параллельными.

Для спецификации разбиения/объединения групп предлагается набор из трех конструкций: *IND* – независимые вычисления, *DIV* – разделяй (divide) и *CON* – властвуй (conquer).

Структуры содержат вложенный блок, в котором могут вызываться другие структуры или вычислительные функции.

```
<load_data();>
con {
        div {
    <base_call();>
  }
}
```

Листинг 2. Пример спецификации параллелизма на языке Frame.

Структура *IND* выполняет вложенный блок в группах из одного процесса. *DIV* – выполняет вложенный блок для всей группы, затем делит группу пополам, и выполняет блок в двух группах половинного размера, и так далее до групп размером два. *CON* действует симметрично *DIV*, начиная с групп размером два, и последовательно объединяя их по парам, пока не получится целая группа.

Внутри вычислительных функций могут использоваться низкоуровневые параллельные операции, но только в пределах текущей группы, заданной описанным выше способом. Существуют специальные конструкции для идентификации группы и положения в группе внутри определений вычислительных функций.

Существующая реализация использует в качестве базового языка определения вычислительных функций C и MPI.

CO_2P_2S (Correct Object-Oriented Pattern-based Programming System)

Разработчики данной системы смотрят несколько под другим углом на идею программирования в терминах ограниченного набора типовых конструкций: они предлагают объектно-ориентированный подход на основе типовых проектных решений (design patterns).

Программисту предлагается набор типовых проектных решений и инструментальное средство, помогающее использовать эти решения при написании прикладных программ. Помощь осуществляется в генерации каркаса (framework) приложения из нескольких связанных классов. Классы каркаса содержат методы-заглушки (hook-method), которые вызываются каркасом; они должны быть переопределены пользователем для решения конкретной задачи. Кроме задания пользовательских функций, каркасы настраиваются с помощью набора параметров-настроек, запрашиваемых у пользователя перед генерацией кода. Генерируется только необходимый минимум

кода, что положительно сказывается на производительности по сравнению с использованием универсальных конструкций с динамической проверкой условий. Могут задаваться общие параметры генерации кода – оптимизация по памяти или скорости, включение отладочного кода. Для настройки каждого из проектных решений, предлагается специальный графический модуль (plugin).

Основным принципом реализации системных каркасов является их корректность. Это означает, что только что сгенерированный каркас в любом варианте настройки представляет собой синтаксически и семантически корректную программу, реализующую какое-либо проектное решение. Эта программа, хоть и задает структуру алгоритма, не делает ничего полезного до тех пор, пока каркас не наполнится необходимой функциональностью[9].

Например, для вычислений на сетке требуется определить два основных метода: вычисление нового значения в узле, на основе значений в соседних, и условие завершения счета; плюс методы инициализации и пост-обработки. Настройками каркаса являются: класс элементов в узлах, топология решетки (замкнутость по размерностям), и шаблон зависимости от значений в соседних узлах.

В системе реализованы несколько последовательных проектных решений, а также параллельные: поиск по дереву, вычисления на двумерной сетке, конвейер, независимый параллелизм, прогонка матрицы.

Поддерживаются модели вычислений с общей и распределенной памятью[11]. В случае общей памяти используются каркасы, основанные на нитях (threads) Java, а в случае распределенной – каркасы на технологии JINI.

В отличие от SKIE в системе нет поддержки вложенности структур. Пользователь может создать несколько каркасов, но их связывание и проблему обеспечения их совместной работы ему придется решать без помощи со стороны системы.

Имеется средство, **MetaCO$_2$P$_2$S**[10], помогающее при создании новых проектных решений для включения в систему. С его помощью создается XML-описание каркаса и графического модуля его параметризации. Кроме этого, разработчик каркаса должен написать код на языке Java, реализующий шаблон каркаса, и специальным образом его проаннотировать.

PASM (Parallel Architectural Skeleton Model).

Целью данной модели[12] является унификация представления типовых алгоритмических структур для обеспечения возможности создания однородной и расширяемой системы программирования. Кроме того, предлагается объектно-ориентированная библиотека типовых алгоритмических структур и механизмов для создания новых ТАС.

Данная модель основывается на том же подходе, что и рассмотренный ранее язык p3l: иерархическая композиция экземпляров типовых алгоритмических структур, но если там механизмы реализации были встроены в систему и скрыты от пользователя специальным языком, здесь предлагается техника чуть более низкого уровня. Пользователь имеет дело с механизмами реализации ТАС, хотя это не означает, что ему приходится писать вручную параллельные алгоритмы. Просто расширяется интерфейс взаимодействия пользовательского кода с окружением. Если в p3l пользовательский код всегда выражен в виде функции с данными на входе и результатами на выходе, здесь нужно общаться с окружением напрямую по установленным протоколам.

Определяется понятие архитектурного шаблона (architectural skeleton), как универсального (generic) строительного блока, содержащего внутри себя реализацию какого-либо параллельного проектного решения. При задании структурных и поведенческих параметров из архитектурного шаблона получается виртуальная машина, с полностью определенной формой и структурой обмена сообщениями или синхронизации. Далее пользователь наполняет виртуальную машину функциональностью и получает параллельный вычислительный модуль. Программа состоит из одного или нескольких взаимодействующих параллельных вычислительных модулей.

Предложенная реализация модели (на C++) активно использует техники объектно-ориентированного программирования. Например, создание виртуальной машины заключается в наследовании от класса архитектурного шаблона, вычислительные модули – объекты классов виртуальных машин.

Виртуальная машина содержит следующие параметры:

- "Представитель" ВМ – код, который взаимодействует с другими ВМ (внешними и вложенными). Здесь же задаются, в частности, вычислительные операции пользователя.
- Вложенные ВМ. Дочерние по отношению к внешней ВМ, соседние (peer) по отношению друг к другу.

- Топология – спецификация взаимодействия соседних ВМ между собой и внешней ВМ.
- Внутренний протокол взаимодействия/синхронизации. Он вместе с топологией задает параллельную вычислительную модель ВМ. Примитивы протокола используются ВМ для взаимодействия с дочерними ВМ, и дочерними ВМ для взаимодействия друг с другом (то есть с соседями).
- Внешний протокол взаимодействия/синхронизации. Примитивы протокола используются ВМ для взаимодействия с соседями и родительской ВМ.

Идея организации взаимодействия заключается в том, что внутренний для данной ВМ протокол является внешним для дочерних ВМ.

Еще одним отличием от p3l является необходимость внесения некоторых алгоритмических дополнений в используемые ТАС (а именно написание "представителей" ВМ). Это в основном код, связанный с получением данных извне (по внешнему протоколу) и распределением их по подструктурам (по внутреннему протоколу). В p3l же вся семантика передачи данных от структуры структуре скрыта от пользователя.

Распределение дерева вычислительных модулей по процессорам осуществляется в динамике при запуске программы, в соответствие с топологиями и иерархией вложенности.

Для облегчения программирования в рамках модели PASM с использованием предложенной авторами библиотеки на C++ создан язык – расширение C++ и транслятор с него в стандартный C++.

Заключение

Рассмотренные в данной работе системы хоть и основываются на одном принципе – повторном использовании типовых структур – во многом различаются: набором абстракций, масштабом, программным обеспечением, а также кругом решаемых задач.

Системы различаются и по расширяемости. SKIE не предоставляет возможностей для добавления новых абстракций, так как основана на намеренно ограниченном структурном языке p3l. Аналогично ограничен и язык Frame. Наоборот, одной из основных задач, стоявших перед создателями системы CO_2P_3S и модели PASM является проблема обеспечения расширяемости.

Наличие или отсутствие расширяемости влияет на круг решаемых задач и на возможности автоматической оптимизации. Если система абстракций зафиксирована, то, во-первых, она достаточно гибкая для расширения применимости, а во-вторых, позволяет создавать модели и инструменты автоматического анализа и преобразования программ, с вытекающей отсюда возможностью оптимизации и переносимостью. Системы, открытые для внесения новых абстракций, хоть и позволяют добавление недостающих частей для решения какой-либо новой задачи, но не могут обеспечить гибкость комбинирования структур, и какой-то общей модели анализа программ. Это демонстрируется на примерах системы SKIE, с одной стороны, и разработок CO_2P_3S и PASM с другой.

В плане дружественности пользователю возможны разные точки зрения. С одной стороны наиболее дружественной представляется система SKIE с её богатым набором программного обеспечения, объединенного в одну среду и визуальными средствами программирования. С другой стороны, для "продвинутых" пользователей более удобным может показаться текстовое программирование в их любимом текстовом редакторе. Кроме того, возможности по расширению модели (например, как в PASM) также можно рассматривать как плюс с точки зрения дружественности пользователю.

Задачей дальнейших исследований в данной области является объединение преимуществ расширяемых и ограниченных систем. Этой цели можно достичь либо созданием универсальной системы абстракций на все случаи жизни, что кажется маловероятным, или же разработкой некоей мета-модели, позволяющей описывать всевозможные ТАС и предоставляющей, во-первых, унифицированный механизм комбинирования ТАС, и, во-вторых, способ анализа программ из призвольных ТАС.

Литература

1. Murray Cole. Algorithmic Skeletons: Structured Management of Parallel Computation // MIT Press, Cambridge, Massachusetts, 1989.
2. Susanna Pelagatti.A methodology for the development and the support of massively parallel programs. PhD thesis // TD-11/93, Departament of Computer Science, University of Pisa (Italy), 1993.
3. B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti and M. Vanneschi, P3L: A structured high level programming language and its

structured support // Concurrency: Practice and Experience vol.7 n.3, pages 225--255, May 1995
4. Andrea Zavanella, Optimizing Skeletal Stream Parallelism on a BSP computer // Proc. of Europar 99 LNCS vol. 1685 Springer Verlag, 1999.
5. Andrea Zavanella. Skel-BSP: Performance Portability for Skeletal Programming // Proceedings of HPCN 2000
6. Andrea Zavanella. The Skel-BSP Global Optimizer: Enhancing Performance Portability in Parallel Programming // Proceedings of Europar 2000.
7. M. Aldinucci, M. Danelutto. Stream parallel skeleton optimization // proc. of the 11th IASTED Intl. Conf. on Parallel and Distributed Computing and Systems, pages 955-962, Cambridge, Massachusetts, USA, Nov. 1999.
8. S. Ciarpaglini, L. Folchi, Design, Implementation and Validation of an MPI template library // http://www.pqe2000.enea.it/home/pqe1/A43-02.zip
9. S. MacDonald. From Patterns to Frameworks to Parallel Programs. Ph.D. Thesis // Department of Computing Science, University of Alberta, 2002.
10. S. Bromling. Meta-programming with Parallel Design Patterns. M.Sc. Thesis // Department of Computing Science, University of Alberta, 2001.
11. Kai Tan. Pattern-based Parallel Programming in a Distributed Memory Environment. M.Sc. thesis // Department of Computing Science, University of Alberta, 2003.
12. Goswami D. Parallel Architectural Skeletons: Re-Usable Building Blocks for Parallel Applications, Ph.D. thesis // University of Waterloo, Ontario, Canada, 2001.
13. M.Cole. Frame: An Imperative Coordination Language for Parallel Programming. Research report EDI-INF-RR-0026 // Division of informatics, University of Edinburgh. 2000
14. M.Hamdan, G.Michaelson & P.King. A scheme for nesting algorithmic skeletons. // Proceedings of 10th International Workshop on Implementation of Functional Languages, University College London, Sept, 1998, pp195-212
15. Adachi, S., Iwasaki, H., Hu, Z. Diff: A Powerful Parallel Skeleton // Proc. 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), CSREA Press, pp.2175-2181, 2000.
16. http://homepages.inf. ed.ac.uk/mic/Skeletons/index.html

Булочникова Н.М., Горицкая В.Ю., Сальников А.Н.

Некоторые аспекты тестирования многопроцессорных систем

В статье рассматриваются тесты, исследующие поведение многопроцессорных систем (в том числе влияние "посторонних" передач на замеряемое время передачи сообщения). Визуализация результатов тестирования осуществляется с помощью разработанных для этой цели приложений, позволяющих отслеживать загруженность процессоров системы, а также состояние коммуникационной среды.

Введение

В современном мире часто возникает проблема эффективного распределения вычислений по процессорам системы. Для решения данной задачи необходимо своевременно получать информацию о текущей загруженности процессоров, а также о состоянии коммуникационной среды.

Иногда причиной недостоверности результатов тестирования многопроцессорных систем является факт его проведения в "идеальных" условиях. Во время запуска теста в системе не происходит других коммуникационных и вычислительных активностей, и поэтому данных оказывается недостаточно для прогнозирования поведения системы в процессе счета реальных, не тестовых, задач. Кроме того, достаточно часто отсутствует адекватная визуализация результатов тестирования, из-за чего дальнейшее изучение закономерностей поведения сети становится чрезвычайно трудоемким для человека занятием.

Действительно, текстовая форма представления информации о производительности сети (файлы с числами) очень неудобна для человеческого восприятия, – ведь если мы имеем дело с большими многопроцессорными системами, то файлы, с которыми придется работать, будут содержать довольно внушительное количество чисел. Не имея возможности графически представить эти данные, мы вряд ли сможем понять характер процессов, происходящих в сети.

Процесс выяснения обстоятельств определённого поведения сети весьма сложен, и на текущий момент времени нет общего механизма, который позволил бы дать ответ на вопрос: какой промежуток времени будет задействован при передаче порции данных определённой длины от одного хоста к другому. При решении этой задачи очень важно обратить внимание на следующие нюансы:

- скорости передачи данных в сети между машинами (хостами) могут быть несимметричны (то есть замеренные

времена передачи одного и того же сообщения в прямую и обратную сторону могут быть различны);

- скорость передачи данных, может сложно зависеть от размера передаваемых данных.

Все вышеперечисленные факты приводят к тому, что человеку, анализирующему поведение сети, приходится держать в голове как минимум четырехмерную модель производительности сети (необходимо помнить, между какими процессорами передается сообщение, размер этого сообщения, время передачи). Дополнительные сложности вносит влияние других обменов сообщениями, происходящих в системе одновременно с основной, измеряемой передачей. Главная же трудность состоит в необходимости выяснения всех деталей поведения каждого элемента сети – без этого картина будет недостаточно полной. К счастью, в течение длительного периода времени сеть ведёт себя достаточно стабильным образом, - это позволяет применять статистические оценки для прогнозирования поведения сети.

Для исследования многопроцессорной системы и дальнейшего планирования вычислений, необходимо знать следующие характеристики поведения сети:

- пропускная способность канала - количество информации, передаваемой между узлами сети в единицу времени (байт в секунду); заметим, что реальная пропускная способность может снижаться программным обеспечением за счет передачи разного рода служебной информации;
- зависимость времени передачи сообщения от размера сообщения;
- влияние "фоновых" передач сообщений на время "целевой" передачи сообщения ("фоновыми" или "шумовыми" сообщениями будем называть передачи, происходящие в многопроцессорной системе помимо передач, время которых измеряем, то есть "целевых" передач):
- "стрессоустойчивость" сети – влияние пиковых нагрузок на работоспособность системы.

Требования к средствам тестирования

Можно сформулировать следующие требования к средствам тестирования сети с точки зрения дальнейшего прогнозирования ее поведения:

- они должны предоставлять необходимую информацию о поведении сети (в том числе о влиянии "фона" на скорость передачи сообщений);
- должна присутствовать возможность варьировать такие параметры тестирования, как длина передаваемого

сообщения, шаг приращения длины сообщения, уровень "шума" и количество "шумовых" процессоров;
- должны быть доступны исходные коды тестов;
- тесты по возможности должны запускаться на большинстве аппаратных платформ, поддерживающих Unix-подобные операционные системы;
- для удобного просмотра и анализа полученной информации должен быть предоставлен удобный для дальнейшего анализа данных графический интерфейс.

Большинство современных средств тестирования многопроцессорных систем, к сожалению, не проводят фоновых замеров, а также в случае коммерческих тестов пользователю не предоставляются исходные тексты программ. Коммерческие тесты обычно весьма дорогостоящие, что не способствует их широкому использованию.

Достаточно часто в современных системах тестирования сетей отсутствуют средства визуализации полученных данных, что существенно затрудняет их дальнейший анализ. А если средства визуализации все-таки встроены в тесты, то они очень громоздки, непереносимы между платформами и в большинстве своем не предназначены для предсказания производительности многопроцессорной системы (визуализируют не те данные, которые, по нашему мнению, были бы полезны для прогнозирования поведения сети).

Цель работы

Целью нашей деятельности было создание набора тестов, выясняющих производительность сети и процессоров, а также переносимых между платформами средств визуализации, которые позволят наглядно представить результаты тестов и относительно легко интерпретировать их. В связи с этим были написаны тесты производительности сети и процессоров. Тесты основаны на интерфейсе передачи сообщений MPI из-за его общеупотребительности и переносимости между платформами. Были созданы также переносимые между платформами средства визуализации данных о поведении сети, реализующие четырехмерную модель представления данных о производительности сети, о которой упоминалось выше (средства визуализации написаны на языке Java2 с использованием графических компонент).

Другие исследования

Хотелось бы упомянуть о других работах, посвященных тестированию производительности мультипроцессорных систем.

В НИВЦ МГУ была разработана система MPI-тестов [2] для определения эффективности программно-аппаратной среды выполнения

параллельных приложений. Эта система позволяет проводить следующие измерения: латентность и скорость пересылок между двумя узлами, пропускная способность сети при сложных обменах по различным логическим топологиям, эффективность основных операций MPI, производительность файл-сервера.

Еще одна система тестирования – Netperf [3]. Она была разработана компанией Hewlett Packard и предназначена для измерения производительности сети. Включает тесты скорости передачи (bandwidth) и задержки (latency) по протоколам TCP и UDP с использованием BSD sockets.

Тесты производительности и загруженности процессоров

В разработанной нами системе тестов производительность процессоров измеряется на эталонном перемножении матриц: на всех процессорах одновременно производится перемножение двух матриц одинакового размера, случайным образом заполненных числами с плавающей точкой. Время работы вычисляется в миллисекундах. Результаты сохраняются в файл как числовой вектор, размерность которого совпадает с числом процессоров.

Для выяснения загруженности процессоров было создано распределенное приложение, получающее информацию о загруженности машины за последние 1, 5 и 15 минут от ядра операционной системы. При этом результат отправляется на выделенную машину, так называемый "монитор".

Тесты сетевых обменов

Тесты сетевых обменов измеряют время передачи сообщений между машинами сети с точностью до микросекунды. По результатам измерений формируется матрица NxN, где N – число задействованных процессоров (элемент (i, j) матрицы соответствует времени передачи сообщения i-м процессом j-му). Для достоверности измерений передача сообщения определенной длины проводится несколько раз (этот параметр может варьироваться пользователем), в качестве результата берется среднее значение полученных данных.

Тестовый набор состоит из 6 программ.

all_to_all – тест основан на одновременной неблокированной передаче данных от каждого к каждому и предназначен для выяснения "стрессоустойчивости" сети.

one_to_one - тест, производящий блокированную передачу от i-го процессора к j-ому. Во время передачи данных остальные процессы (кроме i-го и j-го) "молчат". Замеряется время исполнения функции *MPI_Recv()*, и это время считается временем передачи сообщения от i-го процессора к j-ому. Такая передача производится для всех процессоров, то есть i и j пробегают значения от 1 до n , где n - число

задействованных процессов. При помощи этого теста можно определить пропускную способность каналов связи между процессорами (машинами) при условии, что в сети при этом нет посторонних передач данных, которые могут влиять на производительность сети.

async_one_to_one – тест, осуществляющий неблокированный обмен MPI-сообщениями навстречу друг другу между i-м и j-м процессами. Два процесса одновременно вызывают *MPI_Isend()*, затем с обеих сторон одновременно инициируется *MPI_Irecv()* и *MPI_Wait()*. Время после выполнения *MPI_Wait()* замеряется i-м процессом. При помощи этого теста можно определить, является ли канал между двумя процессорами полнодуплексным. Если канал полудуплексный, то полученные результаты будет в 2 (и более) раза меньше, чем результаты, полученные при помощи теста *one_to_one*.

send_recv_and_recv_send – тест, предназначенный для выяснения усредненной производительности межпроцессорных обменов.

test_noise – то же, что и *async_one_to_one*, но добавлен параметр "шума" (все процессоры разбиваются на две группы: пара процессоров, время передачи сообщения между которыми измеряется, и все остальные; из "остальных" случайным образом выбираются "фоновые" процессоры (их количество задается в командной строке); после этого инициируются неблокирующие передачи; процессоры, не выбранные в качестве "шумовых" и не относящиеся к тем, время передачи между которыми измеряем, "молчат").

test_noise_blocking – тест, измеряющий время блокирующих передач на фоне "шума" (выбор "фоновых" процессоров и уровня "шума" аналогичен тесту *test_noise*).

Результаты тестирования

Тестирование коммуникационной среды проводилось на МВС-1000М (beta) и IBM eServer pSeries 690 (regatta). Ниже приведены некоторые из полученных результатов.

Поведение МВС-1000М при интенсивной одновременной передаче.

На рисунке 1 показана зависимость времени передачи сообщения от длины сообщения на машине МВС-1000М

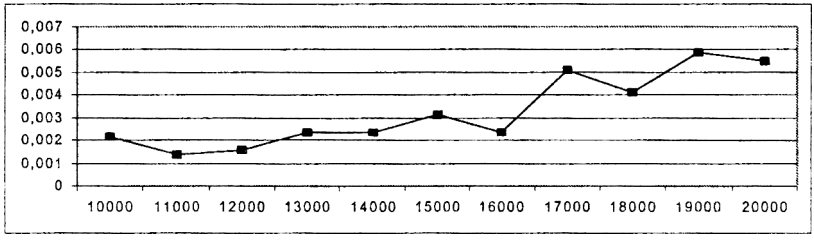


Рис. 1. Тест *all_to_all*, зависимость времени передачи сообщения от длины сообщения для 64 процессоров

(задействованы 64 процессора) на примере теста *all_to_all*. Наблюдается вполне ожидаемая картина: время передачи сообщения растет с увеличением его длины.

Влияние размера "шумового" пакета на время передачи сообщений (МВС-1000М).

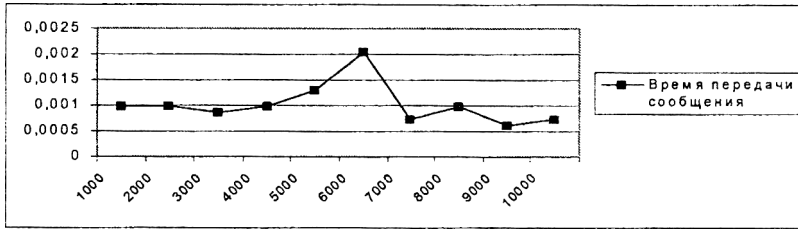


Рис. 2. Тест *test_noise*, зависимость времени от размера "фонового" пакета, длина сообщения – 5000 байт, количество задействованных процессоров - 128

Сравнение времен передач сообщений на фоне "шума" и в его отсутствии.

На рисунке 3 приводится сравнение результатов измерений времени передачи сообщений в присутствии "шума" и без него на машине МВС-1000М. Как видно из графиков, шум несколько замедляет передачу, однако довольно сложно определить характер влияния "фона" на время передачи целевых сообщений.

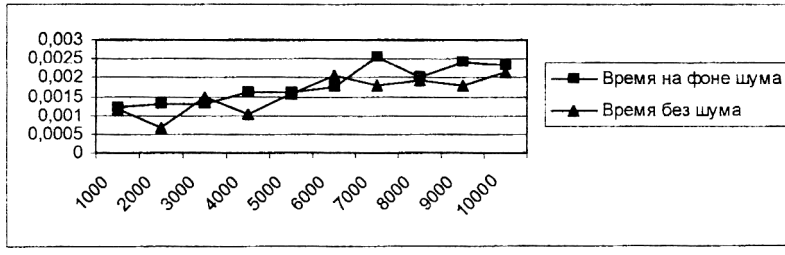


Рис. 3. Тест *test_noise*, размер "фонового" пакета – 1000 байт, количество задействованных процессоров – 64 (МВС-1000М)

Рисунок 4 иллюстрирует сравнение передач с "шумом" и без него на машине regatta. Поскольку regatta представляет собой архитектуру с общей памятью, то графики различаются весьма несущественно.

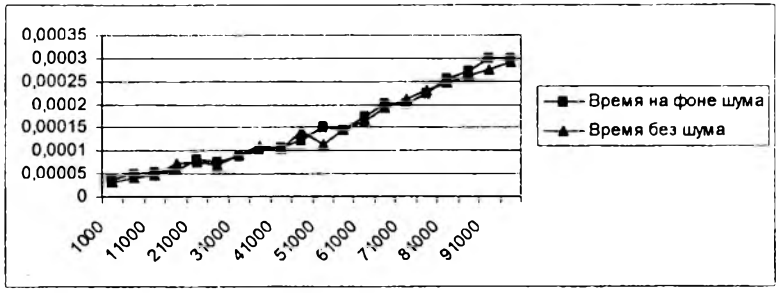


Рис. 4. Тест test_noise_blocking, зависимость времени от размера "фонового" пакета, длина сообщения – 5000 байт, количество задействованных процессоров – 16 (regatta)

Средства визуализации

В связи с вышеперечисленными проблемами были созданы два приложения на языке Java2: одно из них предназначено для визуализации данных о сети (времени передачи сообщений), другое визуализирует данные о хостах (производительность и загруженность).

Визуализация данных о хостах.

Приложение позволяет посмотреть не только пиковую производительность процессоров в вычислительной системе, но и текущую загруженность в процентах и операциях, а также "незанятую производительность" процессора (разность пиковой и текущей производительностей).

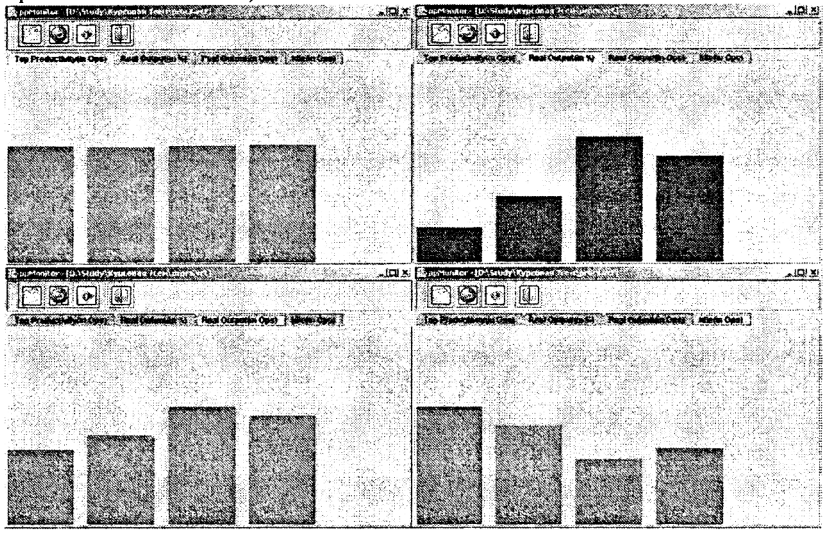


Рис. 5. Визуализация данных о хостах.

Визуализация данных о сети.

Созданы приложения для визуализации сетевой информации, реализующие рассмотренную выше четырехмерную модель представления данных о поведении системы (рис. 6).

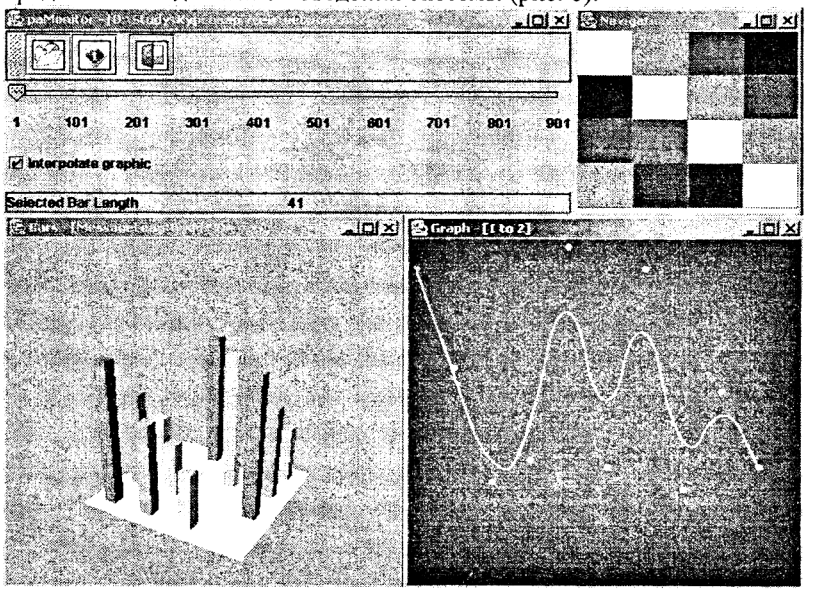


Рис. 6. Визуализация загруженности сети.

По умолчанию загруженность сети изображается программой в виде трехмерной диаграммы, где в горизонтальной плоскости лежат соответственно посылающие и получающие сообщение процессоры. По желанию пользователя может быть установлен режим просмотра результатов в виде поверхности или набора двумерных графиков. Просмотр результатов в виде диаграмм или поверхностей реализован для фиксированной длины посылаемого сообщения и всех процессоров. В программе предусмотрен специальный слайдер, который позволяет перейти к выбранной пользователем длине сообщения. Для фиксированной пары процессоров просмотр результатов реализован в виде двумерного графика (хронология загруженности), отображающего время передачи сообщений разной длины. Процессоры фиксируются при помощи специального навигатора, оформленного в виде таблицы. Цвет столбцов диаграммы, цвет соответствующих областей поверхности и цвет соответствующей ячейки на навигаторе зависят от времени передачи сообщений между процессорами: чем дольше передавалось сообщение, тем насыщеннее цвет визуализируемого объекта. Для визуализации этих объектов используются оттенки серого.

При фиксации двух процессоров на навигаторе столбец, отвечающий этим процессорам, выделяется красным цветом. В момент просмотра хронологии загруженности для выделенной пары процессоров точка, соответствующая текущей длине сообщения, также будет красной.

В качестве иллюстраций представления загруженности сети можно привести рис.7 и 8:

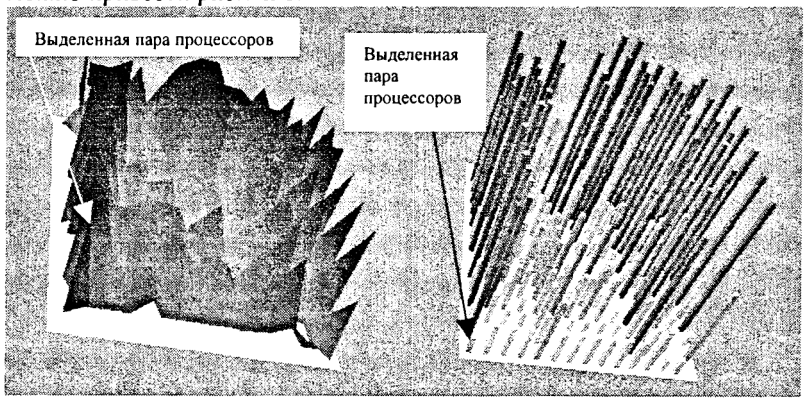


Рис. 7. МВС-1000М, задействовано 15 процессоров, длина передаваемого сообщения 1000 байт

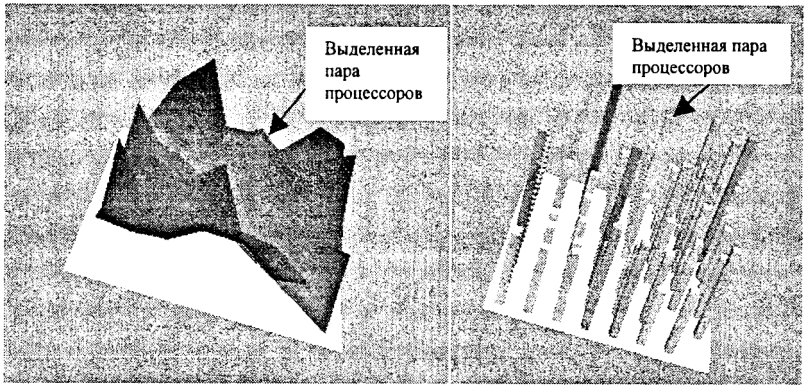


Рис. 8. Regatta, задействовано 8 процессоров, длина передаваемого сообщения 26000 байт.

Выводы

Проблема тестирования производительности сети, безусловно, чрезвычайно сложна и требует дальнейших серьезных исследований.

В процессе написания средств визуализации мы столкнулись с проблемой отсутствия качественных Java-компонент, переносимых между платформами.

На основе проделанной работы можно сделать следующие выводы.

Разработанная система тестирования (состоящая из 6 тестов) делает возможным решение некоторых задач, связанных с выявлением закономерностей поведения сети, в том числе задачи влияния количества фоновых передач на скорость передач целевых сообщений. Были выявлены следующие особенности оборудования МВС-1000М:

- при задании размера шумового пакета в пределах 5000-6000 байт наблюдается заметное замедление скорости передачи целевых сообщений (уточнение информации требует дальнейших исследований);
- при использовании небольшого числа процессоров на МВС-1000М "фоновые" передачи незначительно замедляют скорость передачи целевых сообщений.

На примере regatta показано: для архитектур с общей памятью наличие "фоновых" передач некритично.

Разработанный набор программных средств может быть использован администраторами сетей, а также интегрирован в планировщик распределенных вычислений.

Часть рассмотренных тестов реализована в системе автоматического распараллеливания программ PARUS [1].

В будущем планируется несколько дополнить существующий набор тестов, а именно добавить специальные задержки между передачами для получения картины поведения сети за длительное время. Кроме того, планируется расширить возможности визуализатора (реализовать анимацию, показывающую изменение времени передачи сообщения в зависимости от его размера и т. п.)

Авторы благодарят члена-корреспондента РАН Л.Н. Королева и доцента Н.Н. Попову за консультации и научное руководство в процессе работы над средствами тестирования, а также коллектив МСЦ РАН и администраторов МВС-1000М и машины regatta на факультете ВМиК МГУ.

Исследования проводятся как часть работ по гранту РФФИ 02-07-90130.

Литература

1. Сальников А.Н. Разработка инструментальной системы для динамической балансировки загрузки процессоров и каналов связи. Высокопроизводительные параллельные вычисления на кластерных системах. //Материалы Международного научно-практического семинара. Изд-во Нижегородского университета, 2002г. ISBN 5-85746-681-4 стр. 159-167.
2. http://www.parallel.ru/testmpi/
3. http://www.netperf.org/

Махнычев В. С.

Распараллеливание сложных интеллектуальных задач, основанных на обходе дерева перебора

Введение

Одним из основных и наиболее эффективных алгоритмов обхода дерева перебора является альфа-бета алгоритм, позволяющий значительно сократить число перебираемых вариантов за счет исключения заведомо плохих альтернатив.

Классический алгоритм альфа-бета перебора основан на отсечении части дерева перебора с помощью анализа результатов, полученных для уже обработанных вершин дерева. Таким образом, этот алгоритм изначально предполагает последовательное исполнение. Поэтому распараллеливание этого алгоритма сопряжено со значительными трудностями.

При использовании архитектуры с общей памятью удается реализовать достаточно эффективный параллельный обход дерева перебора с отсечением [1,2], несмотря на некоторую избыточность перебора по сравнению с последовательным альфа-бета алгоритмом. Но существенным недостатком архитектуры с общей памятью является сложность увеличения узлов системы. Поэтому в данной работе предпринята попытка построить эффективную распределенную систему на основе обычной локальной сети, которая легко масштабируется. При этом, так как существующие в архитектурах с общей памятью реализации параллельного альфа-бета перебора основаны на передаче больших объемов данных между процессорами, то в распределенной системе на основе обычной сети приходится искать другие методы эффективного обхода дерева перебора.

Для испытаний реализуемых методов была выбрана игра в шахматы. Этот выбор был произведен по нескольким причинам. Во-первых, дерево шахматной игры достаточно неравномерно разветвлено, причем ветви значительно различаются по высоте: некоторые варианты развития партии могут длиться десятки ходов, а некоторые отсекаются как невыгодные после первого же хода. На дереве с такой сложной структурой недостатки распараллеливания будут заметнее, чем на деревьях с более регулярным строением. Во-вторых, довольно легко получить объективную оценку эффективности построения распределенной системы по силе игры программы. При этом, так как

большинство сильнейших современных шахматных программ используют именно алгоритм альфа-бета перебора (с некоторыми модификациями), то полученная распределенная система, увеличивая мощность этого алгоритма, должна играть в шахматы сильнее, чем современные программы, что само по себе представляет интерес.

Общая схема распределенного перебора

При построении распределенного алгоритма обхода дерева перебора основным требованием является минимизация объемов передаваемых между узлами данных. Минимальных пересылок удается достичь, если разделить дерево перебора на независимые части, каждая из которых обрабатывается отдельным узлом без взаимодействия с остальными узлами. Каждый узел обрабатывает свое поддерево с использованием, например, классического альфа-бета алгоритма или – в случае многопроцессорного узла – с помощью одной из параллельных реализаций альфа-бета алгоритма. В этом случае нужна только передача результатов обработки каждого из поддеревьев головному узлу, который на основе полученных данных выделяет лучшую ветвь дерева.

Таким образом, можно предложить следующую общую схему распределенного обхода дерева. Головной узел системы формирует нужное количество поддеревьев, выбирая их корневые вершины с помощью движения из корневой вершины дерева по различным ветвям на небольшую глубину, и затем рассылает корневые вершины этих поддеревьев остальным узлам. При этом поддеревья должны формироваться с учетом того, чтобы результаты их анализа давали в совокупности наиболее полную картину обо всем дереве перебора в целом.

Проще всего в этом случае формировать поддеревья так, чтобы они в сумме с вершинами дерева, пройденными головным узлом при формировании поддеревьев (то есть в сумме с путями от корневой вершины дерева до корневой вершины каждого поддерева) образовывали все дерево перебора целиком. Например, можно взять из исходного дерева перебора все вершины, отстоящие от корня на заданное расстояние, и принять их за корневые вершины независимых поддеревьев. Однако у этого простого способа есть два серьезных недостатка.

Первый из них заключается в том, что заведомо невыгодные ветви дерева будут исследоваться наравне с выгодными, что приведет к неэффективному использованию вычислительных мощностей. Обычный альфа-бета алгоритм отсекает невыгодные альтернативы на ранних стадиях – как только невыгодность становится очевидной, что позволяет уделять больше времени анализу ветвей, которые с наибольшей вероятностью ведут к выигрышу. В данном же случае мы

не можем отсечь невыгодные варианты, так как, во-первых, ветвь может после более детального рассмотрения стать выгодной, а во-вторых, в случае отсечения потребуется перераспределение заданий между узлами, что вызовет необходимость передачи между узлами больших объемов данных (поскольку отсечения в обычном альфа-бета переборе происходят достаточно часто).

Второй недостаток связан с ограниченностью количества узлов в вычислительной системе.

Следует заметить, что, поскольку в сложных интеллектуальных задачах дерево перебора практически необозримо, то за приемлемое время может быть просмотрена только часть дерева. Чем больше вершин содержит просмотренное алгоритмом поддерево, тем точнее результат, полученный для всего дерева. В большинстве реализаций алгоритм обхода дерева обычно производит несколько итераций, просматривая на каждой итерации дерево на заданную глубину и увеличивая глубину просмотра при переходе к следующей итерации. Количество итераций (и, как следствие, глубина просмотра дерева перебора) ограничено временем, которое отводится на работу алгоритма.

В связи с такой спецификой обхода дерева результаты для каждого из обрабатываемых поддеревьев постоянно уточняются (с ростом глубины просмотра поддеревьев). Следовательно, количество узлов системы должно быть достаточным для того, чтобы обрабатывать все поддеревья одновременно: в противном случае точность результатов для некоторых вершин будет низкой, что отрицательно скажется на точности конечного результата. На примере шахматной партии это означает следующее: не уделив достаточно времени одному из вариантов, мы рискуем просмотреть выигрышную (как для себя, так и для противника) комбинацию.

Но при использовании описанного способа разбиения на поддеревья требования к числу узлов могут оказаться чрезмерными. Например, в середине шахматной партии каждая из сторон может сделать в среднем 50 ходов. То есть указанный способ, двигаясь от корневой вершины дерева на расстояние, равное единице, сформирует около 50 поддеревьев, а если взять расстояние, равное двум, то поддеревьев будет уже около 2500.

Селективная схема распределенного перебора

Естественный путь избавления от описанных недостатков – рассматривать только поддеревья, соответствующие выгодным ветвям исходного дерева. Однако возникает проблема определения того, какие ветви считать выгодными.

В данной работе для решения этой проблемы предлагается следующий алгоритм. На первом шаге из корневой вершины дерева на небольшое время запускается обычный, непараллельный альфа-бета перебор. По результатам этого перебора отбирается несколько лучших ходов. На втором шаге распределенной системой рассматриваются только поддеревья, соответствующие этим лучшим ходам. При этом к каждому из полученных поддеревьев можно сначала применить тот же первый шаг алгоритма, сначала выделив в поддереве несколько лучших направлений непараллельным перебором, а затем уже детально исследовать эти направления с помощью распределенной системы.

Таким образом, на первом шаге выбирается несколько ветвей дерева, а на втором шаге уточняется, какая из этих ветвей наилучшая. Однако в чистом виде эта схема будет работать только в том случае, если выборка хороших ходов, производимая на первом шаге, всегда содержит лучший ход. Если же лучшего хода среди результатов первого шага не оказалось (что вполне вероятно, так как на первом шаге глубина просмотра невелика), то второй шаг лишь уточнит, какой из выбранных первым шагом ходов предпочтительнее, но лучшего хода так и не найдет.

Чтобы избежать такой ситуации, для каждой вершины дерева, где производится первый шаг алгоритма, дополнительно на отдельном узле системы запускается обычный альфа-бета перебор (назовем его *проверяющим*), который работает параллельно со вторым шагом алгоритма. И если для такой вершины проверяющий перебор в какой-то момент получает ход, которого не было среди результатов первого шага алгоритма, то распределенная система начинает анализировать новое поддерево, соответствующее новому лучшему ходу. При этом в случае нехватки узлов можно прекратить анализ наименее перспективного поддерева рассматриваемой вершины. Наибольшие изменения могут потребоваться в случае, если найден новый лучший ход из корневой вершины дерева. Но объем пересылок в любом случае невелик, так как ситуация появления кардинально нового лучшего хода достаточно редка.

Применяя данную схему, можно задействовать практически любое количество узлов. При этом глубина просмотра дерева перебора увеличивается достаточно быстро. Например, если для корневой вершины рассматривать три лучших хода, а для вершин первого уровня – два лучших хода, то потребуется система из десяти узлов, а глубина просмотра возрастет на 2. При использовании описанного ранее простого способа разбиения дерева перебора на поддеревья, для того же увеличения глубины просмотра на 2 потребовалось бы около 2500 узлов.

За экономию узлов приходится платить тем, что поиск новых лучших ветвей дерева происходит с той же скоростью, что и в обычном

последовательном алгоритме перебора. Однако в большинстве практических случаев лучшая ветвь уже присутствует в результатах первого шага алгоритма, и поэтому система в целом производит более качественный анализ дерева и выбор лучшего варианта.

Реализация и результаты испытаний

В рамках данной работы была реализована и испытана селективная схема распределенного перебора. При испытании на 10 узлах использовались приведенные выше параметры: три лучших хода из корневой вершины и по два лучших хода из вершин первого уровня. Испытания проводились на персональных компьютерах различной мощности, объединенных в локальную сеть. Также для испытаний привлекались два узла, подключенные к системе через Интернет. При их использовании заметного снижения качества анализа не произошло.

В текущей реализации для ускорения работы первичный перебор для получения лучших ходов запускается на свободных узлах (если таковые имеются). Это позволяет в некоторой степени распараллелить первый шаг алгоритма.

В случае нехватки свободных узлов предпочтение отдается наиболее перспективным вариантам.

В качестве модуля последовательного альфа-бета перебора, запускаемого на узлах системы, использовалась шахматная программа Shredder – одна из сильнейших на сегодняшний день.

Ниже приведены состояния системы в различные моменты анализа. Каждая анализируемая вершина дерева перебора обозначается желтым кружком. Под кружком отображаются полученные для него промежуточные результаты (оценка позиции). Над кружком – соответствующий ему ход в дереве перебора.

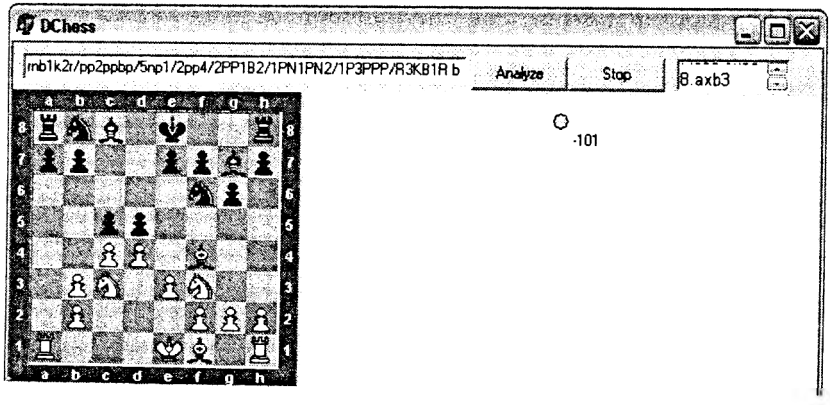


Рис. 1. Начало анализа. Запускается первичный перебор для поиска трех лучших ходов.

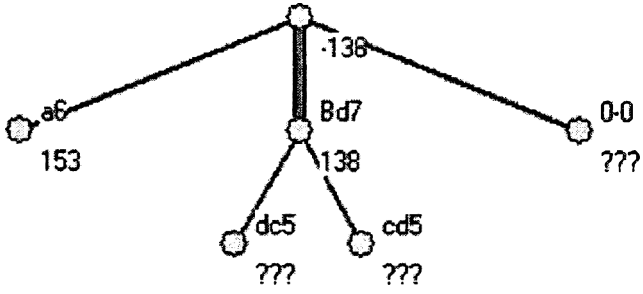


Рис. 2. Первый шаг алгоритма. Промежуточные результаты известны благодаря проверяющим переборам.

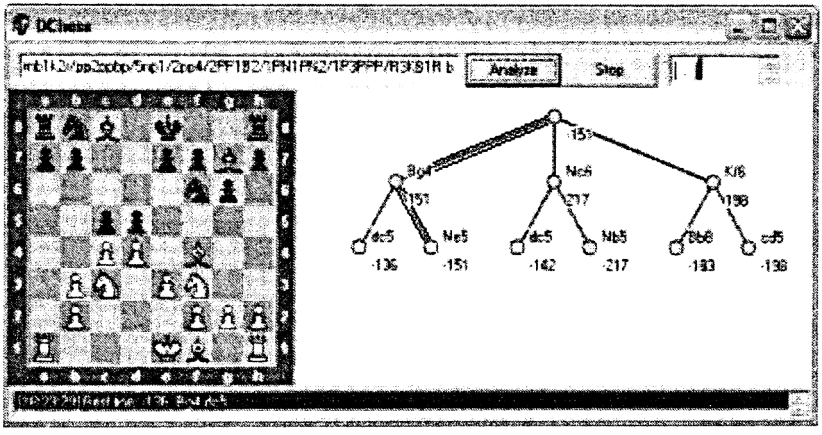


Рис. 3. Все узлы распределенной системы в действии.

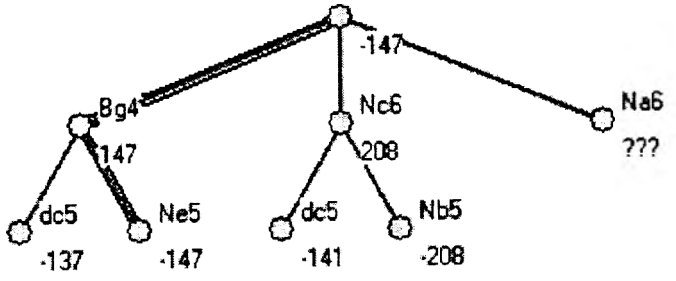


Рис. 4. Появление нового лучшего хода из корневой вершины (Na6).

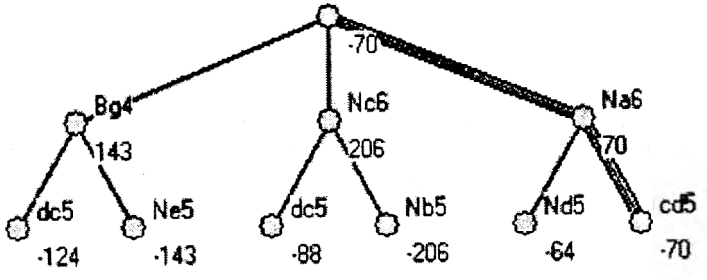


Рис. 5. Результат смены лучшего хода.

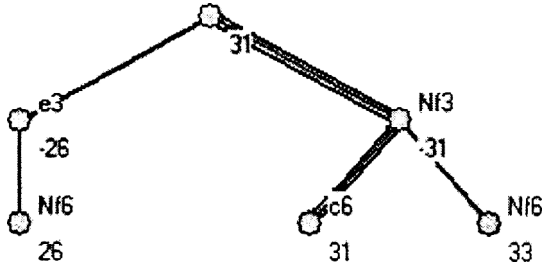


Рис. 6. При нехватке узлов предпочтение отдается наиболее перспективному варианту.

Для оценки эффективности системы необходимо было провести матч между этой системой и какой-либо шахматной программой. Для

этого потребовалось добавить схему контроля времени, с помощью которой принимается решение об окончании анализа позиции и делается найденный к текущему моменту лучший ход. Алгоритм контроля времени был выбран достаточно простой: если в течение фиксированного промежутка времени предлагаемый системой лучший ход не изменялся, то на этом анализ позиции прекращался; в противном случае на анализ отводилось некоторое дополнительное время. Дополнительное время также отводилось в случае нахождения нового лучшего хода одним из проверяющих узлов – с тем, чтобы для нового хода можно было успеть сгенерировать несколько хороших ходов (если после этого хода еще предполагается ветвление и запуск первого шага алгоритма) и получить более или менее качественные результаты анализа соответствующего ходу поддерева.

Поиск лучших ходов первичным перебором занимает несколько секунд, и каждое появление нового лучшего хода также требует дополнительного времени для качественного анализа. Это, а также время реакции модуля Shredder, обуславливают нецелесообразность использования всей системы для быстрого анализа партий или игры в быстрые шахматы. Однако при игре с контролем 30 минут на партию реализованная распределенная система, использующая 8-10 узлов, уверенно выигрывает у той же шахматной программы Shredder, запущенной на двухпроцессорном компьютере, значительно превосходящем по мощности каждый из узлов распределенной системы.

Направления дальнейших исследований

Для улучшения эффективности реализованной системы имеет смысл улучшать первичную выборку хороших ходов, так как если эта выборка не содержит лучшего хода, то выигрыш от использования распределенной системы резко снижается. Улучшить первичную выборку можно двумя способами. Оба способа увеличивают количество рассматриваемых хороших ходов, увеличивая таким образом вероятность наличия среди них лучшего хода, правда, ценой увеличения количества задействованных вычислительных узлов.

Первый способ – использование эвристик для получения дополнительных ходов, помимо тех, которые получатся в результате быстрого перебора. Например, можно рассматривать также невыгодный на первый взгляд, но сильно обостряющий позицию ход в надежде, что этот ход ведет к выигрышной комбинации.

Второй способ – неравномерное (выборочное) увеличение количества рассматриваемых ходов. Например, если лучший вариант для какой-либо вершины дерева часто изменяется, то это означает, что

программа «не уверена» в решении данной позиции, и целесообразно рассмотреть дополнительные варианты.

Напротив, если лучший вариант для какой-либо вершины дерева не изменяется, то может оказаться полезным увеличить глубину ветвления для этого варианта. Возможно, прекратив при этом обработку других вариантов в этой вершине.

Увеличение глубины просмотра на первом шаге алгоритма также можно проводить с использованием дополнительных эвристик.

Для стабильной работы системы на большом количестве компьютеров, не обязательно находящихся в локальной сети, необходимо также повышать устойчивость к отказам отдельных узлов. Кроме того, при передаче данных через Интернет начинают ощутимо сказываться задержки связи. Поэтому имеет смысл классифицировать узлы по надежности и скорости обмена данными с ними, и направлять на узлы задания в соответствии с этой классификацией. Например, задание на первичный перебор с целью получения нескольких хороших ходов должно быть выполнено максимально быстро, так как после выполнения этого задания появится возможность загрузить работой сразу несколько узлов, анализирующих каждый свое поддерево.

Литература

1. http://www.cis.uab.edu/hyatt/search.html
   Hyatt R. The DTS high-performance parallel search tree algorithm. // University of Alabama at Birmingham, 1995.
2. http://supertech.csail.mit.edu/papers/thesis-kuszmaul.pdf
   Kuszmaul B. Multithreading with active messages in tree-search algorithm. // Massachusetts Institute of Technology, 1994.

Раздел IV
Сенсорные сети

Куприянов А. Е.

Защищенная вероятностная маршрутизация в беспроводных сенсорных сетях

Введение

Одна из основных проблем, возникающих в области беспроводных сенсорных сетей, – это обеспечение их безопасности, особенно безопасности сетевого взаимодействия. Различными авторами было предложено большое количество сетевых протоколов для сенсорных сетей. Часть из них является применением в новой области методов, лежащих в основе традиционных сетевых протоколов; другая часть представляет собой результат развития новых идей, отражающих специфику сенсорных сетей. Однако функционирование многих из этих протоколов может быть легко разрушено или изменено в нужном нарушителю направлении. Ситуация усугубляется тем, что из-за присущих сенсорам ограничений уже разработанный сетевой протокол нельзя сделать безопасным постфактум, как это часто происходит в случае традиционных сетей (например, с помощью применения туннелирования). Поэтому сетевой протокол для сенсорных сетей от начала и до конца должен разрабатываться с учетом требований безопасности и ограничений, накладываемых на сенсорные устройства. В настоящей статье проводится анализ безопасности существующих протоколов маршрутизации для сенсорных сетей, а также предлагается разработанный автором сетевой стек протоколов, обладающий свойством защищенности.

Безопасность протоколов маршрутизации для сенсорных сетей

Принципиальная открытость радиопередачи, на использовании которой построена концепция сенсорных сетей, позволяет осуществлять рассматриваемые далее разнообразные атаки на протоколы маршрутизации.

Еще одна важная особенность сенсорной сети с точки зрения ее безопасности – возможность компрометации отдельных сенсоров и

считывание хранящейся в их памяти секретной информации. В силу требуемой дешевизны сенсоров, становится невозможным применение мер по физической защите отдельных сенсоров от компрометации. Скорее, этот вопрос решается размещением избыточного количества сенсоров, заранее предполагая ненадежность их функционирования. Напротив, базовые станции, в силу их малого числа, можно считать доверенными, т.е. защищенными с помощью каких-либо физических мер.

Можно выделить следующие разновидности атак на протоколы маршрутизации (подробнее см. в [1]):

DOS-атака с постановкой активных радиопомех. В рамках данной атаки, нарушитель глушит радиопередачи сенсорной сети с помощью мощного радиопередатчика, работающего в той же полосе частот. Ясно, что такой атаке не может противостоять любой протокол маршрутизации. Однако возможность противостоять такой атаке не следует включать в список требований к протоколу маршрутизации по следующим причинам:

- DOS-атака легко может быть обнаружена базовой станцией и прекращена с помощью физических мер (локализация и устранение источника помех);
- Если исключить хулиганские явления, реализация такой атаки *не является целью* нарушителя. Целью действительного нарушителя будет скорее являться запланированное искажение в нужном направлении картины показаний сенсорной сети в целом или ее отдельной области.

Атака воспроизведением. На сетевом уровне – многие протоколы предусматривают рассылку соседям сигнального пакета. Воспроизводя эти пакеты можно влиять на формирование топологии сети. На уровне передачи данных опасность представляет воспроизведение подтверждения о приеме пакета. С помощью такого воспроизведения можно убедить сенсор, что ошибочная радиопередача была успешно принята.

Туннельная атака. В случае этой атаки пакеты, полученные в одном конце сети, быстро передаются по высокоскоростному каналу связи на другой конец сети и там воспроизводятся. Может быть особенно опасной в сочетании с другими видами атак.

Избирательная маршрутизация. В случае овладения нарушителем несколькими сенсорами, он может управлять ими и осуществлять маршрутизацию только выгодных ему сообщений.

Фальсификация маршрутной информации. Многие протоколы маршрутизации предусматривают принятие решений о построении топологии сети на основе информации, получаемой от соседних сенсоров. Эта информация может включать такие характеристики, как число хопов до базовой станции или стоимость

доставки пакета. Фальсифицируя эту информацию, нарушитель может сделать скомпрометированный сенсор особенно притягательным для соседей, перенаправив через него значительную долю трафика целых областей сети.

Атака «размножением». Предположим, что используется общий ключ канального шифрования на всю сенсорную сеть. Тогда, скомпрометировав один сенсор, нарушитель может его размножить, создав виртуальные сенсоры с разными идентификаторами. Вместо этих сенсоров радиопередачу будет вести нарушитель.

Практически все из существующих протоколов маршрутизации для сенсорных сетей в той или иной степени подвержены перечисленным выше атакам. Среди причин их уязвимости можно выделить следующие:

- *Построение топологии сети на основе информации, полученной от соседних сенсоров.* Фальсификация маршрутной информации позволяет нарушителю менять топологию нужным ему образом.
- *Детерминированная передача сообщений определенному соседнему сенсору*, выбираемому по каким-либо критериям. Нарушитель может оказать влияние на процесс выбора или же подменить выбранный сенсор.
- *Отсутствие аутентификации пакетов или аутентификация на едином для всей сети ключе* позволяет нарушителю, скомпрометировав один сенсор, оказывать существенное влияние на функционирование всей сети в целом.

Основываясь на анализе безопасности существующих протоколов маршрутизации, можно сформулировать требования, которым должен удовлетворять защищенный протокол маршрутизации:

1. Устойчивость к атакам без компрометации сенсоров.
2. Искажение показаний сенсоров и общая производительность сенсорной сети должны ухудшаться не быстрее, чем доля скомпрометированных сенсоров.

Первому требованию легко может удовлетворить любой из перечисленных выше протоколов маршрутизации при условии применения канального шифрования. Второму требованию не удовлетворяет ни один. В случае применения канального шифрования на едином ключе (как в подсистеме безопасности TinySec операционной системы TinyOS), достаточно компрометации единственного сенсора для нарушения работы всей сети. Второе требование будет выполнено, если будут выполнены следующие три условия:

1. Индивидуальная аутентификация пакетов на канальном уровне. В этом случае нарушитель не сможет добавлять в сеть фальшивые пакеты данных от имени других сенсоров.
2. Индивидуальная аутентификация вложенных в пакеты сообщений на сетевом уровне, независимая от аутентификации канального уровня (т.е. с применением других ключей). При выполнении данного условия скомпрометированный сенсор не сможет вставлять фальсифицированные сообщения, якобы сгенерированные другими сенсорами, в исходящие от него пакеты данных.
3. Скомпрометированный сенсор не должен иметь возможность оказывать влияние на процесс маршрутизации других сенсоров. Тогда нарушитель не сможет значительно повлиять на потоки данных, проходящие по сенсорной сети без компрометации большого количества сенсоров.

Таким образом, защищенный протокол маршрутизации должен оставлять нарушителю только один путь для достижения его целей: компрометация статистически значимого числа сенсоров в сенсорной сети. В последующих разделах кратко (ввиду ограниченного объема статьи) описывается разработанный автором стек протоколов, удовлетворяющий указанным требованиям.

Маршрутизация в рамках стека осуществляется на основе концепции *вероятностной маршрутизации*. В традиционных алгоритмах маршрутизация осуществляется на основе построения маршрутных таблиц. Таблицы каким-либо образом создаются и оптимизируются, например, на основе алгоритма кратчайшего пути, и жестко задают маршрут следования пакета от одного узла цепи передачи к другому. Т.е. маршрутизация осуществляется на стороне *отправителя* пакета. В случае широковещательной рассылки, наоборот, передающий сенсор не знает, кто из его соседей примет пакет. Маршрутизация в этом случае реализуется на стороне *получателя* пакета. Сенсор, принявший пакет данных, вероятностным образом принимает решение о его дальнейшей отправке на основе содержащейся в пакете информации и с учетом собственных возможностей.

Стек протоколов состоит из следующих компонент:

- Подсистема безопасности, отвечающая за аутентификацию пересылаемых по радио пакетов данных;
- Протокол доступа к среде передачи W-MAC;
- Протокол вероятностной маршрутизации GPR.

Аутентификация пакетов в сенсорной сети

Существуют два основных ограничения сенсоров, влияющие на структуру подсистемы безопасности, – это их малая вычислительная мощность (8-ми или 16-битный процессор на частоте до 10Мгц) и малый объем оперативной памяти (от 200 байт до нескольких килобайт). Отсюда следует, что для сенсоров неприменима криптография с открытым ключом, предъявляющая повышенные требования к обоим этим параметрам. Что касается симметричной криптографии, то далеко не все симметричные алгоритмы удовлетворяют указанным ограничениям.

В качестве криптографической основы подсистемы безопасности, предлагается использовать блочный шифр RC5 [2]. Данный шифр очень непритязателен с точки зрения вычислительной мощности и требуемого объема памяти, и поэтому хорошо подходит для применения в сенсорных сетях. Для шифрования данный шифр применяется в режиме CBC, а для выработки кодов аутентификации – в режиме CBC-MAC.

В [3] описана схема аутентификации с помощью однонаправленных функций и применение этой схемы в программе SKEY, служащей для аутентификации пользователей компьютерной системы. В [4] на основе этой схемы построен специализированный протокол μTESLA, позволяющий базовой станции осуществлять аутентифицированное широковещание в пределах сенсорной сети. Автором данная схема расширена на аутентификацию всех пересылаемых в сенсорной сети пакетов данных за счет применения специального алгоритма хранения ключей в памяти сенсоров и глобального механизма согласования временных параметров протокола.

Для применения этой схемы требуется, чтобы отправитель и получатель пакета данных были синхронизированы по времени. Для аутентификации пакета отправитель присоединяет к нему MAC-код, созданный с помощью ключа, секретного в этот момент времени, и посылает пакет получателю. Получатель сохраняет у себя полученный пакет до заранее известного момента раскрытия ключа. В установленный момент времени отправитель раскрывает ключ, а получатель с помощью полученного ключа верифицирует MAC-код сохраненного пакета данных. Поскольку на момент получения пакета ключ был секретным, получатель может быть уверен, что пакет был создан отправителем и не был изменен в процессе передачи. В этой

схеме возникает проблема проверки подлинности раскрываемого ключа, решаемая с помощью однонаправленных ключевых цепей.

Однонаправленной ключевой цепью называется следующая конструкция. Отправитель выбирает случайным образом ключ K_N. Далее к нему последовательно применяется односторонняя функция F:

$$K_{N-1} = F(K_N), \quad K_{N-2} = F(K_{N-1}), \ \ldots \ , K_0 = F(K_1)$$

Отправитель сохраняет у себя сгенерированную ключевую цепь, и по доверенному каналу рассылает получателям последнее звено цепи, т.е. K_0. Время разбивается на интервалы. Для аутентификации пакетов, посылаемых в i-м интервале, выполняются следующие действия:

1. Отправитель вычисляет MAC-код на ключе K_i и широковещательно посылает пакет [Data, MAC(K_i, Data)].
2. Получатель хранит у себя ключ одного из предыдущих интервалов K_{i-m}. Также получатель сохраняет у себя все полученные пакеты [Data', MAC'].
3. В конце i-го интервала отправитель широковещательно рассылает ключ K_i.
4. Получатель проверяет, выполняется ли следующее соотношение: $K_{i-m} = F(\ldots F(K_i))$ (m раз).
   Если равенство выполнено, то получатель принимает ключ K_i в качестве нового окончания ключевой цепи отправителя и сохраняет его вместо K_{i-m}. Возможна ситуация, когда получатель не получит пакета с ключом, например, из-за ошибок при передаче. В этом случае ему нужно дождаться раскрытия одного из последующих ключей, верифицировать его, и уже из этого ключа, применяя к нему одностороннюю функцию, он может получить все недостающие ключи с меньшими номерами.
5. Для пакетов, полученных в j-м интервале, $j \in [i-m+1, i]$, получатель проверяет выполнение соотношения MAC(K_j,Data') = MAC'. Если равенство выполнено, соответствующий пакет принимается, иначе отвергается.

Протокол доступа к среде W-MAC

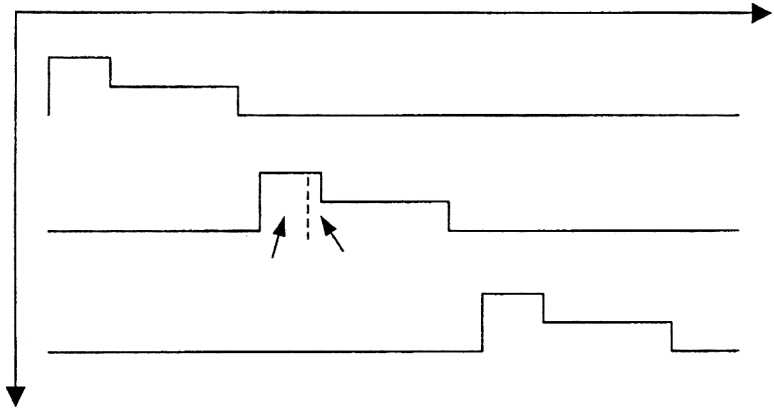
Основная задача, стоящая перед протоколом W-MAC (Wave Media Access Control) – обеспечение сенсорам возможности широковещательной рассылки пакетов. Это означает, что если какой-либо сенсор передает данные, его соседи должны находиться в режиме приема. Ввиду очень ограниченного объема памяти сенсоров, они не в состоянии сохранять большое количество пакетов в ожидании дальнейшей отправки. Поэтому важно таким образом организовать

порядок переключения сенсоров, чтобы вскоре после приема они могли переслать полученные пакеты дальше по цепи передачи.

Способ решения этой задачи основывается на том наблюдении, что сенсорная сеть представляет собой единый комплекс, предназначенный для решения общих задач. В сенсорной сети существуют определенные направления распространения данных (например, от сенсоров к базовой станции), и известна желательная периодичность их распространения. Поэтому в рамках протокола передача сообщений организуется в виде направленных волн.

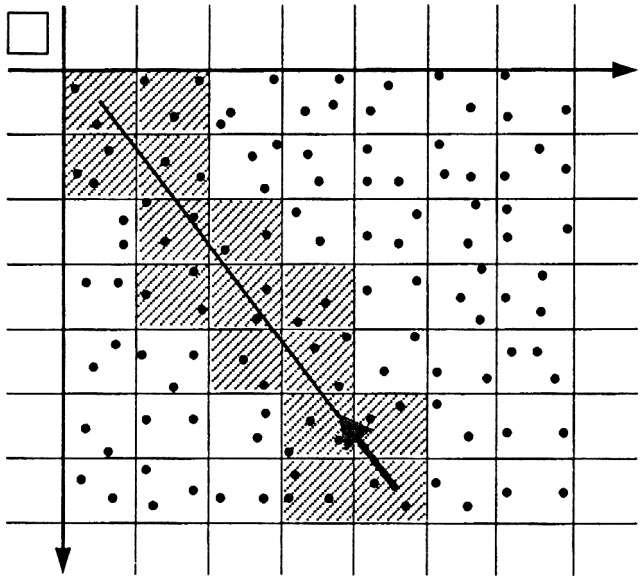
Расписание следования волн составляется базовой станцией на основе требований к периодичности сбора данных, а также размера сети и радиуса влияния радиопередатчиков сенсоров, и широковещательно распространяется по сенсорной сети. Расписание может изменяться со временем в зависимости от режима функционирования сети. Например, в случае отсутствия представляющих интерес событий, период волн может быть существенно увеличен, что снижает время бодрствования сенсоров и позволяет увеличить срок жизни сети. При обнаружении определенных явлений, или же после прихода запроса на сбор данных, соответствующим образом изменяется и график следования волн. В случае локализации явления в определенной области, создаются волны с начальными точками в этой области. Если данные должны быть доставлены в заданную область (например, пользователю, пославшему запрос), соответствующим образом устанавливаются конечные точки.

После получения расписания каждый сенсор действует независимо и включается в момент прохождения через него волны. Вначале он принимает данные от сенсоров, а затем передает часть из полученных данных, а также собственные данные, дальше по волне. В конце выделенного для данной ячейки промежутка времени происходит раскрытие ключей, на которых генерировались коды аутентификации (см. рис. 1).



Область, занимаемая сенсорной сетью, разбивается на дискретные ячейки в соответствии с размерностью протокола (см. рис.2). Это разбиение может быть привязано к физическим координатам сети, а может быть и полностью виртуальным, например на основе достижимости радиопередачи. Получающаяся в результате топология сети в виртуальном пространстве должна удовлетворять следующим требованиям:

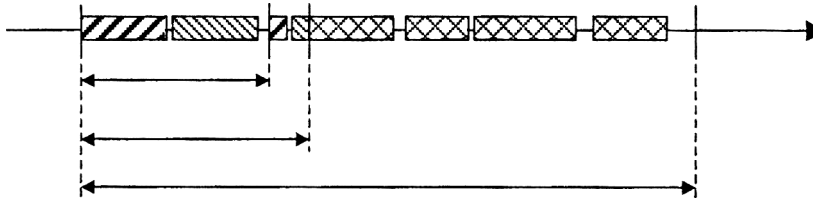
- В каждой ячейке должно находиться примерно одинаковое, небольшое количество сенсоров (1-10);
- Достижимость радиопередачи в виртуальном пространстве должна быть такой, какая получилась бы в результате расположения сенсоров в физическом пространстве с теми же координатами. Как минимум, сенсоры, находящиеся в смежных ячейках, должны слышать радиопередачу друг друга.



Следует отметить, что такое разбиение всегда можно построить за счет увеличения размерности пространства. При этом ячейки, радиопередача которых недостижима друг для друга, разносятся по разным гиперплоскостям. Например, сенсорную сеть, расположенную в здании, можно представить как пучок трехмерных гиперплоскостей в четырехмерном пространстве. Каждая такая гиперплоскость (трехмерное пространство) будет соответствовать своей комнате здания, а места пересечения (плоскости) – дверным проемам. При этом сенсоры,

находящиеся через стену в обычном пространстве (и, следовательно, не слышащие радиопередачу друг друга) в четырехмерном пространстве будут далеко друг от друга.

Время разбивается на эпохи. Для сенсорной сети в целом фиксируется глобальный параметр T_E –продолжительность эпохи (см. рис.3). Этот параметр выбирается на основе требований к реактивности сети, и представляет собой наименьшее время отклика, которое может обеспечить сенсорная сеть. В частности, ни через один сенсор волны не могут проходить чаще, чем один раз за эпоху. Например, для сельского хозяйства T_E может быть установлен равным нескольким минутам, а для систем, требующих более быстрой реакции, таких, как противопожарные – нескольким секундам. Эпохи последовательно нумеруются. В глобальном масштабе времени протокол W-MAC представляет собой TDMA-схему с характерным масштабом времени T_E. Относительно продолжительности эпохи задается расписание следования волн по сенсорной сети. В начале каждой эпохи отводится промежуток времени T_B для радиопередачи базовой станции и срочных передач сенсоров.



Еще один важный параметр протокола, ε – промежуток, не меньший времени переключения радио сенсоров из режима приема в режим передачи. Это время зависит от используемой модели радиопередатчика и может варьироваться в широких пределах. В локальном масштабе W-MAC – это CSMA/CA-протокол с характерной единицей времени ε.

Вероятностная маршрутизация

Протокол маршрутизации GPR (Geographic Probabilistic Routing) расположен на сетевом уровне стека протоколов. Прикладному уровню он предоставляет сервис передачи сообщений в заданную ячейку виртуального пространства сенсорной сети. В свою очередь,

протокол опирается на предоставляемую уровнем передачи данных концепцию направленных волн.

Единицей маршрутизации на этом уровне стека является сообщение. С каждым сообщением связывается приоритет, приближенно отражающий вероятность его доставки в точку назначения. Таким образом, в соответствии с общим вероятностным характером функционирования сенсорной сети, происходит отказ от гарантированной доставки сообщений.

На сетевом уровне поддерживается общий буфер сообщений, полученных от уровня доступа к среде для их маршрутизации, или же с прикладного уровня для отправки. В момент прохождения через сенсор волны, уровень передачи данных сигнализирует сетевому уровню о готовности передавать данные. При этом он сообщает направление прохождения волны. После этого для всех пакетов в буфере рассчитывается их динамический приоритет, который складывается из трех составляющих:

$$P_{dyn} = P_{geo} * P_{packet} * P_{rnd}$$

- Географический приоритет P_{geo} отражает положение данного сенсора в виртуальном пространстве ячеек по отношению к отправителю и получателю сообщения, а также по отношению к волне.
- Приоритет пакета P_{packet} берется из соответствующего поля сообщения.
- Приоритет P_{rnd} задает случайный шум, намеренно вносимый для рандомизации процедуры выбора сообщения.

При расчете географического приоритета учитываются несколько величин, среди которых степень отклонения сообщения от прямого пути между отправителем и получателем и степень направленности текущей волны в сторону получателя.

Сообщение, у которого по результатам расчета оказался наивысший динамический приоритет, передается уровню передачи данных для пересылки. В результате сообщение пересылается от отправителя к получателю по нескольким случайным путям. Следует отметить, что описанный принцип маршрутизации сам по себе является механизмом защиты, поскольку даже в случае компрометации отдельных сенсоров они не могут повлиять на пересылку сообщения по другим путям следования.

Литература

1. C. Karlof, D. Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. // Труды First IEEE International Workshop Sensor Network Protocols and Applications (SNPA'03), май 2003.

2. R.L. Rivest. The RC5 Encryption Algorithm. // Dr Dobb's Journal, v.20, n.1, январь1995.

3. Б. Шнайер. Прикладная криптография. // Изд. Триумф, 2002.

4. A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, D. Culler. SPINS: Security Protocols for Sensor Networks. // Wireless Networks, n.8, 2002.

Куприянов А. Е.

Обзор сетевых протоколов для беспроводных сенсорных сетей

Введение

Беспроводные сенсорные сети являются новым научным направлением, привлекающим все большее внимание в связи с широкими возможностями, предоставляемыми сенсорными сетями для различного рода приложений: от мониторинга окружающей среды до управления сложными системами и процессами. Сенсор объединяет в малом объеме (как ожидается в скором будущем – 1 мм^3) различные датчики (температуры, давления, освещенности, ускорения, химического состава, и т. д.), вычислительный модуль (микроконтроллер), радио приемопередатчик (трансивер), источник питания и различные вспомогательные компоненты, такие, как таймер и аналогово-цифровые преобразователи.

Сенсорная сеть состоит из большого числа (от сотен до десятков тысяч) сенсоров, которые распределяются в представляющей интерес области физического мира, собирают требуемую информацию и либо локально ее обрабатывают, либо передают для обработки базовой станции. Каждый сенсор в отдельности обладает слабыми возможностями, но объединенные вместе они представляют собой принципиально новую сетевую и вычислительную парадигму [1]. В ближайшем будущем сенсорные сети проникнут практически во все области деятельности человека, многократно расширяя его возможности по исследованию и контролю окружающего мира.

Классификация сенсорных сетей

Для более полного понимания особенностей сенсорных сетей необходимо провести классификацию решаемых ими задач. Рассматриваемые здесь классы приложений сенсорных сетей частично основываются на работе [2].

Сбор данных об окружающей среде. Классическим примером приложений этого класса является распределение сенсоров исследователем в некотором представляющем для него интерес районе. Сенсоры постоянно собирают данные и периодически отправляют их по

сети к базовой станции для хранения и обработки. Базовая станция может представлять собой обычный персональный компьютер, или же быть шлюзом, позволяющим изучать область исследования через сеть Интернет. К этому классу относятся применения в различных областях естественных наук (биология, геология), сельском хозяйстве (наблюдения за посевами, контроль режима температуры и влажности), строительстве (контроль состояния несущих конструкций зданий), транспорте (контроль состояния дорожных покрытий, железнодорожного полотна и линий электропередач).

Системы экстренного реагирования. Сенсорные сети этого класса размещаются в некотором районе и контролируют его для обнаружения некоторого аномального поведения. Главное отличие этого класса сетей от предыдущего заключается в том, что сенсоры не собирают никаких данных. Однако в случае обнаружения отслеживаемой аномалии они должны очень быстро передать сигнал тревоги. К этому классу относятся системы безопасности, противопожарные и военные системы.

Слежение за объектами. Еще одна область применения сенсорных сетей – отслеживание перемещения выделенного объекта в области, занятой сенсорной сетью. Для слежения за объектом достаточно прикрепить к нему один сенсор. В этом случае сенсоры никак не взаимодействуют с физическим миром, а только друг с другом, отслеживая радиопередачи мобильных сенсоров и вычисляя их текущее расположение. Сенсорные сети этого класса характеризуются постоянно меняющейся топологией сети. Сюда относятся системы управления складскими запасами, транспортными потоками, производственными процессами, различного рода «умные пространства» (Smart Spaces), обеспечивающие комфортные условия своим обитателям, а также военные системы.

Короткоживущие сети. Отличительная особенность этого класса – короткий срок жизни сенсоров. К данному классу относятся, например, противопожарная сенсорная сеть после обнаружения возгорания, военные сенсорные сети во время боевых действий, и, в общем случае, любые сенсорные сети, работающие в условиях чрезвычайной ситуации. Для короткоживущих сетей, в отличие от остальных классов, важным является не снижение энергопотребления для увеличения срока жизни сети, а скоординированная работа для обеспечения максимальной пропускной способности.

Гибридные сети. В общем случае реальная сенсорная сеть будет соединять в себе характеристики нескольких из вышеперечисленных классов, причем в разных режимах функционирования одна и та же сеть будет относиться к разным классам.

Общесистемные характеристики сенсорных сетей

С точки зрения оценки функционирования сенсорной сети как единого целого, важны следующие ее характеристики:

Стоимость установки и эксплуатации является важнейшим из преимуществ сенсорной сети по сравнению с традиционными проводными системами. Для ее установки не требуется протягивать километры проводов – достаточно разместить требуемое количество сенсоров, которые самостоятельно образуют необходимую конфигурацию сети в соответствии с окружающими условиями. В дальнейшем, если потребуется, всегда можно увеличить плотность сети путем установки дополнительного количества сенсоров. Для обеспечения этих преимуществ отдельный сенсор должен стоить менее $1 – цель, которая, как ожидается, будет достигнута в ближайшие годы.

Срок жизни – еще одна характеристика, имеющая принципиальное значение для сенсорных сетей. Для большинства сетей срок жизни должен составлять не менее нескольких лет. Для удовлетворения этого требования любой протокол для сенсорных сетей должен разрабатываться с оптимизацией по энергопотреблению.

Время отклика для различных классов сенсорных сетей может составлять от секунд до часов. При этом требования к времени отклика могут изменяться в зависимости от режима функционирования сети.

Безопасность относится к необходимым характеристикам сенсорной сети. Учитывая открытый характер радиопередачи, при отсутствии должного обеспечения безопасности функционирование сети может быть легко нарушено.

Индивидуальные характеристики сенсоров

Микроконтроллер сенсора – как правило 8-ми или 16-битный, поддерживает очень ограниченное множество команд, причем только над целыми числами. Современные микроконтроллеры объединяют в одном чипе процессор, оперативную память, аналогово-цифровые преобразователи и цифровой ввод/вывод. Микроконтроллеры, в зависимости от производителя, потребляют от 0.250 до 2.5 мА на 1 МГц в рабочем режиме (при напряжении 3В) и от 1 до 50 мкА в режиме сна.

Оперативная память, которую может иметь сенсор, составляет от 200 байт до нескольких килобайт. Оперативная память является очень дорогостоящим ресурсом, поэтому важно обеспечить способность сенсора работать при минимальном ее объеме.

Радиопередатчик является с энергетической точки зрения самым дорогостоящим элементом сенсора. В то время как ток,

потребляемый большинством других элементов сенсора при напряжении в 3В, составляет не более 1-2 мА, типичный радиопередатчик потребляет около 10-50 мА. Еще одна важная характеристика радиопередатчиков, применяемых в сенсорах, отмечаемая в [2], заключается в том, что они потребляют примерно одинаковое количество энергии как в режиме передачи, так и в режиме приема (в соотношении около 1.5 / 1). Таким образом, энергозатраты на радиопередачу доминируют в энергетическом бюджете сенсора. Для увеличения срока жизни сенсорной сети необходимо строить сетевые протоколы таким образом, чтобы минимизировать время радиопередачи и приема. Важная характеристика радиопередатчиков – время включения (до 2мСек), а также время перевода передатчика из режима передачи в режим приема и наоборот (до 250мкСек). Радиус передачи типичного передатчика составляет до 50 метров.

Датчики, устанавливаемые на сенсоре, могут измерять освещенность, температуру, влажность, давление, магнитные поля, ускорение, звук, содержание в воздухе различных веществ и другие величины. Датчики, в зависимости от типа, потребляют от 0.5 до 2 мА. Время включения датчиков составляет от 30 мкСек до 400 мСек.

Особенности функционирования сенсорной сети

Ненадежность функционирования отдельных сенсоров. В работах [3, 4] рассматриваются итоги длительного (в течение 4 месяцев) функционирования сенсорной сети. Отмечается высокая вероятность отказа как сенсора в целом, так и отдельных датчиков. Уже через 60 дней отказали 90% размещенных сенсоров.

Ненадежность радиопередачи. Вероятность передачи пакета данных по радиоканалу без ошибок от одного сенсора к другому не превышает 50-70%. В работе [5] показано, что для корректной передачи пакета данных в протоколах маршрутизации на основе построения остовного дерева, в среднем требуется не менее 2 повторов передачи.

Неравномерность, нестабильность и асимметричность радиопередачи. Как показано в [5], распределение вероятности приема пакета очень сильно зависит от направления передачи. В результате, в одних направлениях вероятность приема на больших расстояниях может быть существенно больше, чем в других направлениях при меньших расстояниях. Более того, такая картина распределения вероятности приема сильно меняется с течением времени. Также в значительном числе случаев (около 20%) отмечалась асимметричность радиопередачи, когда в одном направлении вероятность приема существенно отличалась от вероятности приема в обратном направлении.

Протоколы маршрутизации

Типичный способ организации сенсорной сети заключается в выделении одной или нескольких так называемых *базовых станций*, т.е. вычислительных узлов, предназначенных для сбора и анализа данных сенсорной сети. По сравнению с обычными сенсорами базовая станция обладает гораздо большей вычислительной мощностью, мощным радиопередатчиком, и для нее отсутствуют ограничения по энергопотреблению. Соответственно, выделяется три основных вида передачи сообщений в рамках сенсорной сети:

- *От базовой станции к сенсорам* – управляющие команды и запросы на сбор данных;
- *Локальный обмен сообщениями между соседними сенсорами* – применяется для самоорганизации сенсорной сети;
- *От сенсоров к базовой станции* – собранные данные или информация о наблюдаемых событиях.

Для передачи первого вида сообщений можно использовать широковещание базовой станции по всей сенсорной сети. Для второго вида – широковещание сенсоров своим соседям. Поэтому наибольшую сложность представляет последний вид передачи, для которого приходится использовать многозвенную (multihop) передачу от сенсора к сенсору.

Различными авторами было предложено большое количество протоколов маршрутизации для сенсорных сетей. Многие из них являются развитием протоколов для традиционных сетей. Однако появились и совершенно новые классы протоколов, в частности протоколы с передачей сообщения по нескольким путям (multipath routing) и географическая маршрутизация.

Протоколы на основе построения остовного дерева.

К этому классу относятся протоколы, построенные на основе известного алгоритма кратчайшего пути Дейкстры (Shortest Path First, SPF) [6]. Поскольку в рассматриваемом виде коммуникаций единственная точка назначения – базовая станция, то этот алгоритм строит остовное дерево с базовой станцией в качестве корня и листьями – сенсорами. Различные протоколы отличаются способом построения этого дерева, и способом изменения его с течением времени. Представителями этого класса протоколов являются:

TinyOS Beaconing [7]. Этот протокол используется в операционной системе TinyOS и заключается в следующем. Базовая станция регулярно рассылает соседним с ней сенсорам специальный сигнальный пакет (beacon). Сенсор, получивший сигнальный пакет, помечает своего соседа, от которого он получил пакет, в качестве

родителя, и широковещательно рассылает пакет дальше. Так строится остовное дерево от базовой станции к каждому из сенсоров. Когда сенсору требуется передать пакет базовой станции, он пересылает его своему родителю.

Minimum Cost Forwarding [8]. Данный протокол аналогичен (с учетом замены терминов) алгоритму дистанционно-векторной маршрутизации в сети Интернет. Каждый сенсор периодически распространяет специальный пакет, в котором объявляет *стоимость* передачи сообщения от него к базовой станции. Сенсор слышит объявления своих соседей, и выбирает из них минимальную стоимость, с учетом стоимости передачи к сенсору, отправившему пакет. Сенсор, предложивший минимальную стоимость, выбирается в качестве родителя.

Протоколы с передачей сообщения по нескольким путям.

В описаниях протоколов этого класса используется специальная терминология. Принимается, что основной объект наблюдения сенсорной сети – это *события*. Событием называется некоторое явление (например, повышение температуры), наблюдаемое одним из сенсоров. Наблюдающий событие сенсор называется *источником (source)* события. Другие субъекты (сенсоры или базовая станция) могут выражать *интерес* к определенному типу событий, в этом случае они называются *стоком (sink)* события. Во всех протоколах этого класса происходит установление нескольких путей, ведущих от источника к стоку события.

Directed Diffusion [9]. В рамках этого протокола базовая станция (сток) выражает интерес в виде запроса, сходного с запросом к базе данных, например:

- Тип – четвероногое животное
- Интервал – 20 мсек
- Продолжительность – 10 сек
- Область – прямоугольник [100,100] – [200,400]

Этот запрос широковещательно распространяется по сенсорной сети. Каждый сенсор, получивший этот запрос, устанавливает *градиент* в направлении того сенсора, от которого запрос был получен. Каждый градиент имеет определенную величину. Источник события, удовлетворяющего запросу, посылает его по соответствующему градиенту. Базовая станция, получив ответ на запрос, может *усилить* градиенты тех сенсоров, от которых ответ пришел первым, и *ослабить* градиенты остальных сенсоров. Это усиление градиента распространяется к источнику события.

Rumor Routing [10]. Когда источник наблюдает некоторое событие, он посылает агента, который случайным образом блуждает по сенсорной сети. Агент переносит следующую информацию: перечень

наблюдаемых событий, следующий узел пути к этим событиям и поле времени жизни (TTL, Time To Live). Сенсор, получивший агента, обновляет собственную информацию о наблюдаемых в сети событиях, уменьшает поле TTL на единицу, и отправляет агента дальше, если TTL>0. Если базовую станцию интересует информация о каком-то событии, она отправляет агента со списком требуемых событий. Установление пути от источника к стоку события происходит в тот момент, когда агент базовой станции попадает к сенсору, через который ранее прошел агент с требуемыми событиями.

Другие примеры протоколов этого типа описаны в [11], [12].

Географическая маршрутизация.

Поскольку сенсорные сети тесно связаны с физическим миром, появляется возможность использовать в качестве сетевых адресов географические координаты сенсоров. На основе развития этой идеи построены следующие протоколы:

Greedy Perimeter Stateless Routing (GPSR) [13]. В рамках этого протокола для передачи пакета в точку с заданными географическими координатами используется «жадный» алгоритм. В соответствии с этим алгоритмом в качестве следующего узла для передачи выбирается сенсор, находящийся наиболее близко к точке назначения среди всех соседей сенсора-маршрутизатора. В случае отсутствия узла с такими координатами, следующий узел выбирается по правилу правой руки, как наиболее близкий к точке назначения сосед, находящийся справа от сенсора-отправителя.

Geographic and Energy Aware Routing (GEAR) [14]. Предыдущий протокол обладает следующим недостатком: каждый из сенсоров всегда будет выбирать один и тот же следующий узел для передачи сообщения базовой станции. Это может привести к более быстрому исчерпанию энергии одних сенсоров по сравнению с другими. Протокол GEAR решает эту проблему производя взвешенный выбор следующего узла на основе информации о его географических координатах и оставшейся у него энергии.

Ряд авторов рассматривает проблему установления координат сенсоров для целей географической маршрутизации при отсутствии точной информации об их местоположении. В работах [15] и [16] для этого используется построение виртуальной координатной системы на основе информации о достижимости радиопередачи.

Протоколы доступа к среде

Ключевым требованием, предъявляемым к MAC-протоколам в сенсорных сетях, является их энергетическая эффективность. Для выполнения этого требования MAC-протокол должен в максимальной

степени устранять факторы, повышающие энергопотребление. К таким факторам относятся:

- *Коллизии.* Если в процессе радиопередачи одного сенсора одновременно происходила радиопередача другого сенсора, то оба пакета будут повреждены, что приведет к необходимости повторной передачи.
- *Холостое прослушивание* происходит, когда сенсор ожидает приема данных, которые не были переданы.

Можно выделить следующие классы MAC-протоколов:

Контроль несущей / избежание коллизий

Схема CSMA/CA являются наиболее простой разновидностью MAC-протокола. Она применяется, например, в операционной системе TinyOS [17]. Каждый сенсор постоянно прослушивает радиоканал на предмет радиопередачи. В случае обнаружения стартового символа радиостек переходит в режим приема. Перед передачей осуществляется задержка на некоторую случайную величину, в течение которой прослушивается радиоканал. Если в конце этого промежутка не было принято ни одного бита в течение 12 тиков таймера, то начинается радиопередача. Иначе делается еще одна задержка на другую случайную величину.

Разделение по времени

В соответствии с подходом TDMA, каким-либо образом устанавливается расписание, согласно которому каждый из сенсоров периодически переводит радио из активного состояния в состояние сна с низким энергопотреблением. На этом принципе построены следующие протоколы:

S-MAC [18]. Каждый сенсор самостоятельно выбирает свое расписание сна и бодрствования и периодически широковещательно рассылает его соседям. В случае если сенсор еще не имеет собственного расписания, он в течение определенного промежутка времени прослушивает радиоэфир на предмет передачи расписания соседних сенсоров. Если он получил такое расписание, он также начинает ему следовать. Если нет – выбирает самостоятельно. Если он получил разные расписания от нескольких соседних сенсоров, он следует всем полученным расписаниям.

Для избежания коллизий в данном протоколе применяется схема RTS/CTS (Request to Send / Clear to Send). В соответствии с ней перед отправкой большого пакета данных сенсор посылает запрос на передачу RTS и должен дождаться получения разрешения на передачу CTS.

T-MAC [19]. Данный протокол является усовершенствованием протокола S-MAC. В соответствии с ним, сенсор может

преждевременно закончить период бодрствования в случае, если за определенный промежуток времени не произошло *активирующего события*. К активирующим событиям относятся передача управляющего пакета SYNC, прием данных, наблюдаемая радиопередача другого сенсора, окончание передачи данных сенсора, а также знание об окончании передачи другого сенсора (в результате предшествующего приема пакетов RTS/CTS).

Разделение по частоте

Еще одним способом организации MAC-протокола является разнесение коммуникации разных сенсоров по частоте (FDMA, Frequency Division Multiple Access). Для этого необходима аппаратная поддержка радиопередатчиком изменения частоты. К протоколам этого типа относится ***SMACS*** [20]. В соответствии с ним каждая пара сенсоров выбирает собственное расписание коммуникаций, а также случайным образом выбирает одну частоту из доступной полосы частот. Поскольку коммуникация между разными парами сенсоров будет происходить на разных частотах, практически снимается проблема коллизий. Однако следует отметить, что в этом случае полностью пропадают и все преимущества радиопередачи, когда одну передачу могут принять сразу несколько сенсоров.

Литература

1. D. Estrin, R. Govindan, J. Heidemann, S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. // Труды ACM/IEEE International Conference on Mobile Computing and networking (MobiCom'99), август 1999.

2. J. Hill. System Architecture for Networked Sensors. // Диссертация на степень доктора философии, Университет Калифорнии в Беркли, весна 2003.

3. A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson. Wireless Sensor Networks for Habitat Monitoring. // Труды First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), сентябрь 2002.

4. R. Szewczyk, J. Polastre, A. Mainwaring, D. Culler. Lessons From A Sensor Network Expedition. // Труды European Workshop on Sensor Networks (EWSN'04), март 2004.

5. A. Woo, T. Tong, D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. // Труды конференции SenSys'03, ноябрь 2003.

6. В. Столлингс. Современные компьютерные сети. // Изд. Питер, 2003.

7. P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, D. Culler. The Emergence of Networking Abstractions and Techniques in TinyOS. // Труды симпозиума Networked Systems Design and Implementations (NSDI'04), март 2004.

8. F. Ye, A. Chen, S.Lu, L. Zhang. A scalable solution to minimum cost forwarding in large sensor networks. // Труды Tenth International Conference on Computer Communications and Networks, 2001.

9. C. Intanagonwiwat, R. Govindan, D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. // Труды Sixth Annual International Conference on Mobile Computing and Networks (MobiCom'00), август 2000.

10. D. Braginsky, D. Estrin. Rumor Routing Algorithm For Sensor Networks. Труды First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), сентябрь 2002.

11. D. Ganesan, R. Govindan, S. Shenker, D. Estrin. Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks. // Mobile Computing and Communications Review, Volume 1, Number 2, 2000.

12. S. De, C. Qiao, H. Wu. Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks. // Труды конференции WCNC'03, 2003.

13. B. Karp, H.T. Kung. GPSR: Greedy Perimeter stateless Routing for Wireless Networks. // Труды Sixth Annual International Conference on Mobile Computing and Networks (MobiCom'00), август 2000.

14. Y. Yu, R. Govindan, D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. // Технический отчет, 2001.

15. A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, I. Stoica. Gegraphic Routing without Location Information. // Труды Ninth Annual International Conference on Mobile Computing and Networks (MobiCom'03), сентябрь 2003.

16. J. Newsome, D. Song. GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks Without Geographic Information. // Труды конференции SenSys'03, ноябрь 2003.

17. N. Lee, P. Levis, J. Hill. Mica High Speed Radio Stack. // Документация проекта TinyOS, 2002.

18. W. Ye, J. Heidemann, D.Estrin. An energy-efficient MAC protocol for wireless sensor networks. // Труды 21st Conference of the IEEE Computer and Communications Societies (Infocom'02), июнь 2002.

19. T. van Dam, K. Langendoen. An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. // Труды конференции SenSys'03, ноябрь 2003.

20. K. Sohrabi, J. Gao, V. Ailawadhi, G. Pottie. Protocols for Self-Organization of a Wireless Sensor Network. // IEEE Personal Communications, октябрь 2000.

Раздел V
Инструментальные средства и сообщения

Герасимов С.В. Машечкин И.В. Петровский М.И. Розинкин А.Н.

Мультиагентная архитектура для Machine Learning систем фильтрации спама масштаба предприятия.[4]

В статье рассматривается система предотвращения массовых рассылок электронной почты уровня предприятия, использующая для классификации писем алгоритмы машинного обучения. Преимущество таких алгоритмов перед традиционными методами обнаружения спама – более высокий уровень обнаружения, независимость от внешних баз знаний, персонализация. Тем не менее, до сих пор интеллектуальные методы не получили широкого распространения в системах обнаружения спама уровня предприятия по причине их недостаточной устойчивости к ложно-положительным ошибкам и повышенных требований к вычислительной мощности. Рассматриваемая система предлагает решение этих недостатков с помощью достаточно быстрого и точного алгоритма, устойчивого к ошибкам, а также мультиагентной архитектуры, позволяющей повысить производительность и упросить масштабируемость. В статье представлены результаты эксперимента, доказывающие превосходство предложенного подхода над наиболее распространенным в настоящее время методом Байеса.

Введение

По различным оценкам от 40 до 80% всех электронных сообщений в интернет являются спамом[1,2]. Несанкционированные рассылки наносят очевидный ущерб – это реальные материальные потери компаний и пользователей сети. Например, убытки только крупных IT-компаний в США 2003 году от спама оценивается в $20.5 миллиарда[3]. Излишняя нагрузка на сети и оборудование, потраченное время на

[4] Работа поддерживается грантом РФФИ 03-01-00745

сортировку и удаление писем, потраченные деньги на оплату трафика – это неполный перечень проблем, которые приносит спам.

В настоящее время разработаны различные методы, направленные на предотвращение получения спама пользователями. Их можно разделить на две группы: традиционные, использующие фиксированный набор правил или сигнатур для фильтрации спама, и адаптивные, основанные на методах статистики и искусственного интеллекта.

Многие традиционные методы основаны на использовании различный типов черных списков с адресами спамеров и адресами почтовых серверов, которые используются для рассылки спама[4,5]. Также к традиционным[5] относятся методы, использующие базы знаний ключевых слов, правил и сигнатур писем со спамом. Такие базы знаний составляются вручную экспертами и требуют регулярного обновления.

Системы, основанные на таких методах как правило имеют невысокий уровень обнаружения спама (порядка 60-70%). Другой их недостаток это необходимость постоянно поддерживать базы знаний в консистентном состоянии. Система привязана и зависима от оперативности конкретного сервиса – провайдера обновлений. В период между появлением нового спама и обновления базы знаний система остается незащищенной. Традиционные методы как правило не персонифицированы, т.е. не учитывают особенности переписки конкретного пользователя, что также отрицательно влияет на их точность.

Системы, использующие черные списки почтовых серверов получили довольно широкое распространение благодаря простоте их использования. Тем не менее, в последнее время появляется достаточно много сообщений, о том что такие системы имеют большое количество ложно-положительных ошибок из-за того, что в списки попадают целые диапазоны адресов интернет-провайдеров. Политика, по которой строятся эти списки иногда непоследовательна и важно понимать, что используя такую систему пользователь целиком полагается на провайдера, который ее поддерживает.

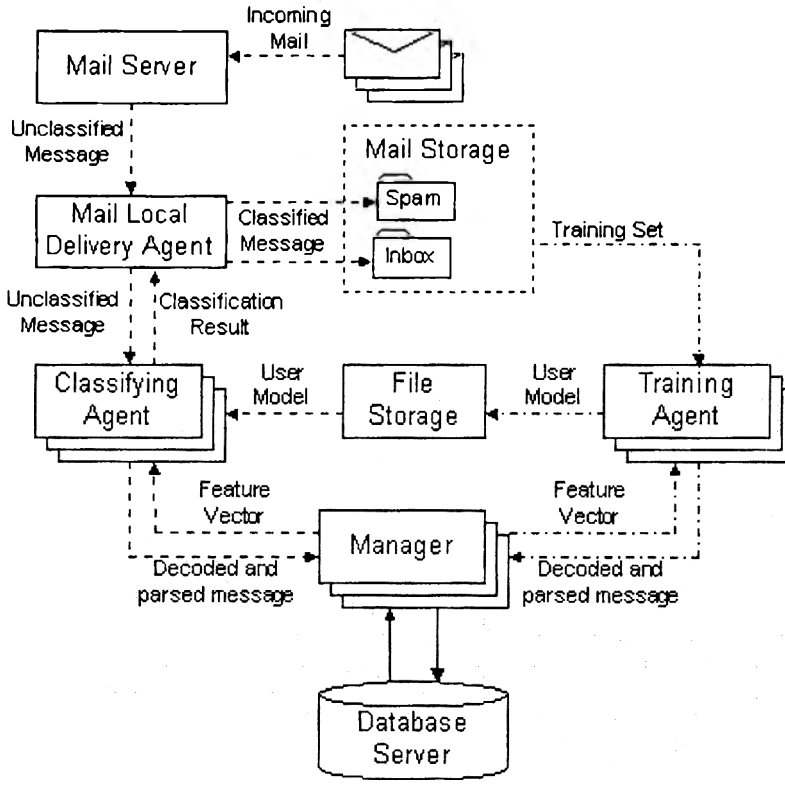
Относительно новым направлением среди методов обнаружения спама являются интеллектуальные методы. Такие методы используют алгоритмы машинного обучения и статистические алгоритмы. Интеллектуальные методы обладают набором преимуществ по сравнению с классическими. Такие методы автономны, независимы от внешних баз знаний, не требуют регулярного их обновления. Являются по сути многоязычными, независимыми от естественного языка. Способны обучатся на новых видах спама с минимальным участием пользователя. Наибольшее распространение из интеллектуальных

методов в настоящее время получил метод Байеса. [7,8]. Он реализован и успешно применяется в некоторых системах обнаружения спама[9,10,11,12].
Тем не менее, несмотря на свою эффективность интеллектуальные персонифицированные методы практически не используются в системах обнаружения спама на уровне почтового сервера масштаба предприятия по причине недостаточной устойчивости к ложно-положительным ошибкам и повышенной требовательности к производительности аппаратной платформы.
Целью данного исследования было предложить комплексное решение для системы классификации почтовых сообщений серверного уровня, основанное на интеллектуальных методах анализа сообщений. Предложенное решение должно обладать такими преимуществами интеллектуальных методов как персонификация, автономность, высокая точность обнаружения спама при низком количестве ложно-положительных ошибок. В то же время система классификации сообщений должна обеспечивать производительность на уровне почтового сервера масштаба предприятия.

Решение

Предложенный архитектурно-программный комплекс основывается с одной стороны на интеллектуальном алгоритме классификации, что позволяет достичь необходимых характеристик качества, и с другой стороны на мультиагентной серверной архитектуре, что обеспечивает необходимую производительность.

Для решения задачи классификации был предложен статистический алгоритм, относящийся к классу методов опорных векторов [13,14].
Для алгоритма была подобрана kernel-функция, а также другие параметры, обеспечивающие высокий уровень точности и приемлемую скорость работы.
Отдельная задача для статистического метода – выбор оптимального пространства признаков над которым будет работать алгоритм [15]. Метод опорных векторов не так чувствителен к размерности этого пространства, как большинство других методов машинного обучения.
Для разработанного алгоритма был построен эвристический метод отбора признаков, приемлемый с точки зрения качества классификации, скорости работы и размерности пространства признаков.



Сравнение предложенного алгоритма на тестовых наборах с наиболее распространенным в настоящее время из интеллектуальных алгоритмов методом Байеса показало, что разработанный алгоритм обладает большей точностью, особенно в части ложно-положительных ошибок. Сравнение и анализ результатов приводится ниже.

На основе разработанного алгоритма была спроектирована и реализована система обнаружения спама серверного уровня. Данное решение должно удовлетворять следующим основным требованиям:

- централизованное решение, учитывающее персональные особенности переписки каждого из пользователей
- потенциальная независимость от платформы реализации
- возможность масштабирования

- приемлемый уровень безопасности и privacy

Учитывая предъявляемые требования для реализации проекта была выбрана мультиагентная архитектура.
Центральным коммуникационным узлом всей системы является один или несколько веб-серверов, с помощью которых осуществляется взаимодействие с пользователем, и взаимодействие между различными агентами системы. Благодаря возможности масштабируемости веб-сервер не является узким местом системы.
Обучающий агент – компонент системы, который выполняет анализ писем пользователя, строит на их основе модель, которая затем используется для классификации новых писем.
Обучающий агент может быть реализован для различных хранилищ писем, например, на основе IMAP на централизованном сервере или персональный агент, использующий для обучения письма из персональных папок пользователя, откачанных по POP3 и хранящихся на рабочей станции пользователя.
Классифицирующий агент получает новые письма от почтового сервера, используя персональную модель, построенную для конкретного пользователя, классифицирует их, и возвращает результат серверу.
Общие структуры данных, необходимые для построения персональных моделей классификации, а также различная вспомогательная информация хранятся в базе данных.
Таким образом, для расширения функциональности такой системы фильтрации спама за счет подключения новых почтовых серверов или добавления поддержки специфических почтовых клиентов достаточно реализовать нового классифицирующего или обучающего агента. Это позволяет создать единую хорошо масштабируемую корпоративную систему фильтрации спама, объединяющую разнородные клиенты и почтовые сервера, работающие в рамках компании.

Используя предложенную архитектуру в качестве базовой, была разработана система классификации почтовых сообщений серверного уровня.
Эта версия системы была опробована на вычислительной системе следующей конфигурации: операционная система RedHat Linux 9.0, почтовый сервер Sendmail 8.2 или Exim 4.34, агент локальной доставки Procmail 3.22.
Обучающий агент по запросу пользователя либо по заданному расписанию анализирует письма пользователя, получая их от почтового сервера по протоколу IMAP. Затем строит модель, и хранит ее в файловой системе.

Классифицирующий агент инициализируется агентом локальной доставки почты. В этой конфигурации использовался Procmail. Он передает классифицирующему агенту тело письма. Тот в свою очередь анализирует его, и возвращает результат агенту доставки. После этого агент доставки во-первых добавляет в заголовок письма его статус (спам/не спам) и вероятность принадлежности к спаму, во-вторых опционально письма классифицированные как спам перекладывает в соответствующую папку пользователя.

Таким образом, пользователь, читая почту по протоколу IMAP в папке Inbox видит только нормальные письма. Письма со спамом не удаляются, а сохраняются в отдельной папке, также доступной пользователю по протоколу IMAP. После дообучения или переобучения пользователь может очистить папку, хранящую спам, для экономии дискового пространства.

Таким образом, одну из проблем, свойственных интеллектуальным методам – их высокую ресурсоемкость, предложенное решение позволяет решить благодаря использованию мультиагентной архитектуры, позволяющей легко масштабировать систему, а кроме того объединять разнородные почтовые сервера и клиенты, работающие на предприятии.

Эксперимент

В настоящее время среди интеллектуальных методов обнаружения спама наибольшее распространение получил метод Байеса. Именно он был выбран для эксперимента, доказывающего эффективность разработанного алгоритма. Сравнение с методом Байеса на эталонных тестовых наборах писем. Для сравнения был выбран алгоритм Байеса, реализованный в свободно распространяемом фильтре спама SpamAssassin[9]. Для эксперимента SpamAssassin был настроен так, чтобы в классификации принимал участие только метод Байеса. Эксперимент проводился на нескольких тестовых наборах писем. Основной целью данного эксперимента было показать преимущество по точности предложенного алгоритма над наиболее распространенным интеллектуальным алгоритмом, основанным на методе Байеса.

Для сравнения разработанного алгоритма и алгоритма Байеса, реализованного в почтовом фильтре SpamAssassin использовались два эталонных тестовых набора писем.

LingSpam Corpus[16]

Изначально имеется 4 версии набора LingSpam. Для своего тестирования использовалась наиболее общая версия набора – bare. Письма в наборе представлены как текст, содержат только лишь тело (body) и тему (subject), но не содержат заголовков (header). Размер набора – 2893 сообщения.
Далее, набор разбит случайным образом на 10 равных частей, каждая из которых состоит в среднем из 290 сообщений, из них 240 нормальных и 50 спам-сообщений. Тестируя алгоритмы на этом наборе писем, в каждой итерации использовались 9 частей набора для тренировки и один оставшийся для тестирования. Таким образом, для этого набора было выполнено 10 различных тестов.
SpamAssassin Corpus[17]
Для тестирования использовался один из наборов писем, представленных на сайте SpamAssassin. Письма представлены полностью, все заголовки были сохранены. Набор состоит из трех частей:

- spam - 500 сообщений, содержащих спам;
- easy_ham - 2500 нормальных сообщений, обычно хорошо распознаваемых фильтрами спама как нормальные;
- hard_ham - 250 нормальных сообщений, но сильно похожих на сообщения, содержащие спам.

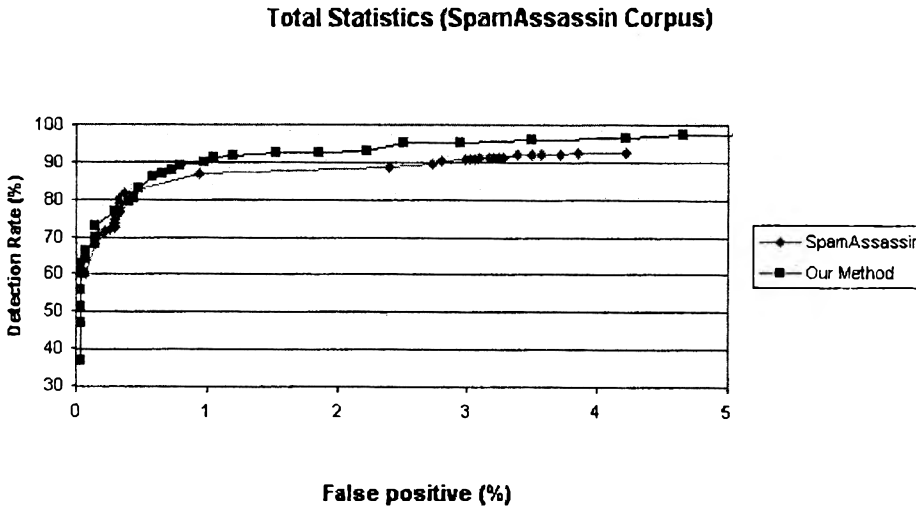
Для тестирования три указанные раздела набора были разбиты на 5 равных частей. Соответственно, каждая часть содержит 100 сообщений спама, 500 легко обнаруживаемых нормальных сообщений и 50 трудно обнаруживаемых нормальных сообщений. Далее, 4 части набора использовались для тренировки и 1 оставшийся набор для тестирования. В итоге было проведено 5 различных тестов с этим набором.

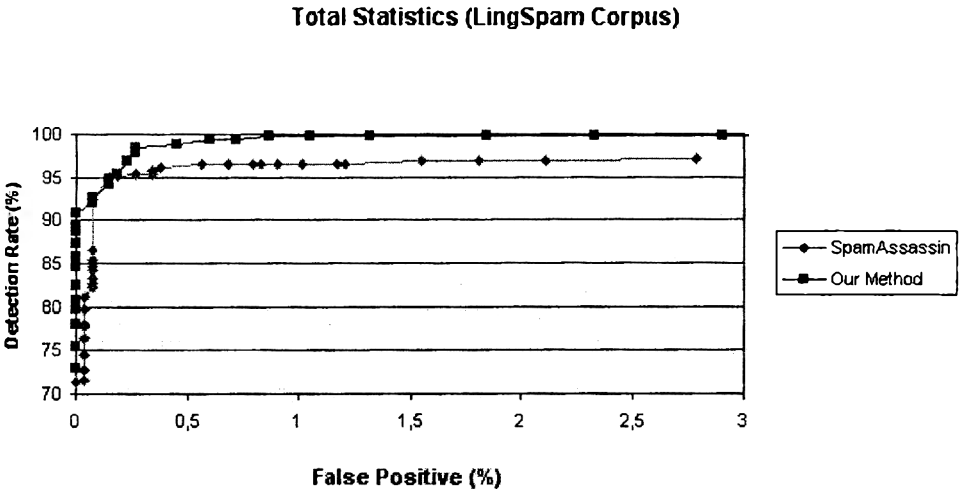
Для анализа результатов этого эксперимента была применена стандартная методика сравнения алгоритмов, имеющих ошибки первого и второго рода с помощью характеристических кривых (ROC curves, Receiver Operating Characteristic curves).

Характеристическая кривая – это кривая, отражающая взаимосвязь ошибок. Ось абсцисс графика кривой определяет значения коэффициента ложных положительных ошибок (false positive rate), ось ординат – значения коэффициента обнаружения (detection rate). Каждому результату работы алгоритма на графике соответствует одна точка. При этом в зависимости от параметров алгоритма получается множество точек. При изменении параметров алгоритма получается кривая, соединяющую точки (0,0) и (1,1).
При прочих равных условиях лучшим алгоритмом будет тот, чей график на этой плоскости расположен «выше». Чем ближе график к точке (0,1), тем более качественно работает алгоритм.

Для каждого тестового набора результаты нескольких тестов были усреднены и по полученным значениям были построены ROC-кривые. Можно видеть, что для обоих тестовых наборов, как для LingSpam так и для SpamAssassin corpus, ROC-кривая для разработанного алгоритма находится выше кривой для алгоритма Naïve-Bayes.
Это означает, что этот алгоритм превосходит алгоритм Naïve-Bayes, реализованный в фильтре почты SpamAssassin по обоим, используемым критериям качества. При любом фиксированном коэффициенте обнаружения (Detection rate) коэффициент ложных положительных ошибок (False positive rate) для разработанного алгоритма ниже. Также и наоборот – при любом фиксированном коэффициенте ложных положительных ошибок коэффициент обнаружения для этого алгоритма выше.

Приведенные результаты тестирования алгоритмов позволяют утверждать, что в целом на используемых тестовых наборах данных экспериментальный алгоритм показал лучшие результаты по сравнению с алгоритмом Naïve-Bayes, реализованным в фильтре почты SpamAssassin.





Анализ тестов показываст, что алгоритм Naïve-Bayes от SpamAssassin настроен прежде всего на минимизацию ложно-положительных ошибок, платой за это является то, что ни на одном тестовом наборе он не достигает 100% эффективности. Разработанный алгоритм в этом смысле является более сбалансированным, имеется возможность в зависимости от потребностей пользователя, изменяя параметры алгоритма, добиваться либо лучшего качества при обнаружении спама, либо минимизировать ошибки false positive.

Заключение

Очевидные недостатки систем, основанных на традиционных методах классификации спама – такие как низкий уровень обнаружения, необходимость регулярно обновлять базы знаний из внешних источников, отсутствие персонализации. Новые интеллектуальные методы с успехом решают эти проблемы. Но использование этих методов в системах классификации почты на уровне сервера почтовых сообщений не распространено из-за их ресурсоемкости недостаточной точности в части ложно-положительных ошибок.

Разработанное архитектурно-программное решение предлагает с одной стороны достаточно точный и быстрый алгоритм, качество классификации которого лучше чем у наиболее распространенного в

настоящее время метода Байеса. Это позволяет добиться более высокого уровня обнаружения.

С другой стороны – благодаря использованию мультиагентной архитектуры, которая позволяет легко масштабировать систему, решается проблема производительности, свойственной интеллектуальным методам, а также дается возможность строить единую корпоративную интеллектуальную систему фильтрации спама в масштабах предприятия, объединяющую различные почтовые системы, работающие в компании. Пилотная программная реализация предложенного подхода, а также ее экспериментальная проверка показала, что предложенный подход превосходит существующие интеллектуальные и традиционные методы фильтрации спама, что позволяет рассматривать его как перспективную платформу для построения систем фильтрации спама масштаба предприятия.

Литература

[1] Spam Filter Software Review, eCatapult, Inc. http://www.spamfilterreview.com

[2] Email Systems, Email Systems Ltd. http://www.emailsystems.com

[3] The Radicati Group, Inc. The Messaging Technology Report, 2003 http://www.radicati.com

[4] ORDB.org, Open Relay Database http://www.ordb.org

[5] MAPS - Mail Abuse Prevention System, LLC http://www.mailabuse.com

[6] Yiming Yang, An Evaluation of Statistical Approaches to Text Categorization, 1999

[7] Paul Graham, A Plan For Spam, 2002 http://www.paulgraham.com/spam.html

[8] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk email,1998

[9] The Apache SpamAssassin Project http://spamassassin.apache.org

[10] POPFile – Automatic Email Classification http://popfile.sourceforge.net

[11] Spam Inspector, GIANT Company Software, Inc. http://www.spaminspector.com

[12] SpamPal - Mail Classification Program http://www.spampal.org

[13] V. Vapnik, Statistical Learning Theory. Wiley, 1998

[14] B. Scholkopf, A. Smola, Learning with kernels: Support Vector Machines,
Regularization, Optimization and Beyond. MIT Press, Cambridge, 2000
[15] Yiming Yang and Jan O. Pedersen, A comparative study on feature selection in text categorization, 1997
[16] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, and Constantine D. Spyropoulos. An evaluation of Naïve-Bayesian anti-spam filtering. In Proceedings of the workshop on Machine Learning in the New Information Age, 2000
http://www.aueb.gr/users/ion/data/lingspam_public.tar.gz
[17] SpamAssassin Public Corpus,
http://spamassassin.apache.org/publiccorpus/

Муравлев В.В.

Поиск дефектов программного кода: методы и средства

Введение

В настоящее время программисты часто сталкиваются с «унаследованными» информационными системами. Эти системы могут доставаться в наследство от сторонней команды разработки, или быть разработаны этой же командой, но в другом составе. Как правило, такая система попадает в руки специалистов, потому что владельцу (пользователю) информационной системы потребовалось внести какие-то изменения в текущую функциональность системы или добавить новые возможности.

Учитывая динамичное развитие ИТ индустрии и, как правило, немалую текучесть кадров в программистских фирмах, вероятность того, что через относительно небольшой промежуток времени поддержку и доработку созданной системы будут осуществлять другие люди, достаточно высока. Также зачастую можно наблюдать процесс разработки системы в условиях постоянных изменений требований заказчика. Нередки ситуации, когда в процессе создания системы в соответствии с первоначальными потребностями заказчика, выясняется, что некоторые из них потеряли свою актуальность в силу изменившихся внешних условий; некоторые пожелания, которые не представляли особого интереса для заказчика, приобрели высокий приоритет и пр.

Эти факторы заставляют предъявлять особые требования к создаваемой системе, а также к процессу ее модификации. Помимо традиционных требований, таких как надежность, удобство в использовании, защищенность и пр., становится актуальным качество проектирования и программного кода системы. Система должна быть проста в понимании и модификации. Это обеспечивает снижение затрат (временных, ресурсных) на включение в команду разрабочиков нового программиста и на внесение новой (или изменение существующей) функциональности системы.

В силу этого, особую значимость обретают такие понятия, как качество кода, его читаемость, качество проектирования. Дать формальное определение качественного или некачественного кода,

дизайна системы практически невозможно. Эти понятия субъективны, и то, что с точки зрения одного программиста является нормальным кодом, для другого может быть совершенно неприемлемо. Однако, существует немало эвристических правил, соблюдая которые, программист может сберечь немало времени и ресурсов не только себе, но и своим коллегам, которые будут пользоваться результатами его труда. Для того, чтобы сохранить код в системе ясным и понятным, программист должен постоянно выполнять ревизию кода на предмет возможности переработки кода – рефакторинга.

Рефакторинг и дефекты кода

Определение рефакторинга

Есть несколько определений понятия «рефакторинг», которые зависят от контекста использования этого слова.

Рефакторинг (Refactoring) — изменение во внутренней структуре программного обеспечения, имеющее целью облегчить понимание его работы и упросить модификацию, не затрагивая наблюдаемого поведения [FOWL].

Т.е. это может быть добавление новой переменной или удаление существующей, добавление нового класса, перемещение функций между классами и т.п. Иногда для обозначения вышеопределенного значения используют словосочетание «метод рефакторинга». Мартин Фаулер в своей книге [FOWL] составил каталог методов рефакторинга, проведя их классификацию по области применения. Также используют слово «рефакторинг» для обозначения процесса:

Применение рефакторинга (Refactor) — изменение структуры программного обеспечения путем выполнение ряда рефакторингов, не затрагивая его поведения. [FOWL].

Рефакторинг может (и должен) проводиться как в разрабатываемой системе, так и для «унаследованной» системы. Для создаваемой системы процесс проведения рефакторинга не должен выделяться в отдельную стадию проекта – рефакторинг должен проводиться постоянно на протяжении всего процесса создания программного кода системы. В случае, когда к программистам попадает «унаследованная» система, код которой сложен и запутан, рефакторинг способен сделать его простым, четким и ясным.

Дефекты кода

М.Фаулер советует применять рефакторинг в следующих случаях: при добавлении нового функционала в систему, при разборе кода группой разработчиков, при появлении повторяющихся участков кода и при исправлении ошибок. При этом программисту предлагается руководствоваться понятием дефекта (в оригинале «запаха») кода [FOWL, XP] - некими эвристическими или субъективными представлениями о том, чего не стоит делать. Если программист, просматривая код, образно говоря, чувствует некий «запах» (иначе говоря, у него возникает ощущение того, что код следовало бы написать иначе) – ему следует провести рефакторинг. В [FOWL] приводится список наиболее популярных и известных дефектов кода, составленный совместно М.Фаулером и К. Беком, например таких, как дублирующийся код, длинный метод, завистливые функции и т.д. Для каждого дефекта существует ряд методов рефакторинга, которые его устраняют.

Выбор конкретного метода (и решение о ликвидации дефекта как такового) может зависеть от множества причин и в конечном счете остается за программистом. Последствия невнимания к таким дефектам могут быть катастрофическими для всего проекта. В результате небрежности (или неопытности) программистов программный код системы может стать запутанным, сложным, насыщенным дублирующимися фрагментами и, следовательно, слабо пригодным к модификации и поддержке.

Поэтому ревизию и переработку кода следует проводить в течение всего времени создания и изменения информационной системы. Очевидно, что для современных информационных систем, программный код которых может состоять из многих тысяч строк, ручная ревизия и внесение исправлений – нереальная перспектива. Необходимы автоматизированные инструменты анализа и внесения изменений.

Инструментальная поддержка

На данный момент в области выполнения методов рефакторинга сделано уже немало. М.Фаулер в [FOWL] приводит для каждого метода рефакторинга из своего каталога (всего 70 методов рефакторинга) пошаговые инструкции по выполнению. W. Opdyke в своей работе [Opdyke] рассматривает методы рефакторинга с точки зрения сохранения внешнего поведения программы. D. Roberts в своей работе [Roberts] исследует зависимости между методами рефакторинга для проверки возможности выполнения цепочек методов рефакторинга.

Существуют промышленные инструменты, поддерживающие выполнение ряда методов рефакторинга, таких как перемещение членов класса (полей и методов) в другие классы, преобразование иерархии классов, переименование переменных и классов, выделение/встраивание методов и др. Примерами таких инструментов могут служить следующие среды разработки:

- IntelliJ IDEA http://www.intellij.com
- Borland JBuilder http://www.borland.com
- Together Control Center http://www.together.com
- Eclipse http://www.eclipse.org
- Refactoring Browser http://www.refbr.com

Все эти средства широко используются в промышленной разработке программного обеспечения. Встроенные средства поддержки рефакторинга оказывают существенную помощь разработчикам. Также существуют отдельные инструментальные средства для выполнения рефакторинга, такие, как RefactorIT (http://www.refactorit.com) и Jfactor (http://www.jfactor.com), но широкого распространения они не получили. На примере этих средств можно видеть, что одним из основных требований, предъявляемых к подобным инструментам является тесная интеграция с существующими популярными средами разработки ПО. Наличие подобных средств «под рукой» у программиста стимулирует его к постоянному их использованию (точнее - не препятствует). Вышеперечисленные инструменты обеспечивают хорошую поддержку и автоматизацию выполнения методов рефакторинга, но аналитическая составляющая в них развита недостаточно. Фактически, ни одно из этих средств не поддерживает автоматический (или полу-автоматический) анализ программного кода на предмет поиска дефектов кода. Между тем, в направлении автоматизации такого поиска уже сделаны некоторые шаги. В следующих разделах будет рассказано о существующих подходах к этой проблеме.

Методы поиска дефектов кода

В этом разделе мы рассмотрим различные способы идентификации дефектов программного кода путем анализа исходного программного кода системы. Но прежде мы опишем более детально сами дефекты. Как уже указывалось выше, понятие дефекта кода («запаха») субъективно. Точное определение как частного случая дефекта, так и понятия в целом, дать очень сложно. То, что является недостатком для одного программиста, для другого может быть совершенно обычным куском программного кода. Но недостатки кода, описанные в [FOWL] основаны не только на личных предпочтениях

автора – это концентрация опыта предыдущих поколений программистов. Некоторые из них могут показаться незначительными, но постоянное пренебрежение к нарушению этих правил способно привести программный код системы в неудовлетворительное состояние. Опишем некоторые из них.

Одним из наиболее распространенных дефектов является дублирующийся код [FOWL, стр. 86]. Одинаковые кодирующие структуры в разных местах должны быть сведены в одну – это несомненно улучшит читаемость кода, повысит устойчивость кода к изменениям и увеличит степень повторного использования кода. Как правило, для этого достаточно создать метод и перенести в него необходимую функциональность (если повторяющиеся куски встречаются, скажем, в пределах одного класса). Другим не менее распространенным недостатком кода являются чересчур длинные тела методов. Длинные методы очень трудны для чтения и понимания. Избежать этого можно путем декомпозиции большого метода на несколько мелких. Это улучшит читаемость метода и повысит возможность повторного использования кода за счет новых методов. Существует негласное правило, согласно которому тело метода должно полностью умещаться на экране (кроме строк с обработкой исключительных ситуаций).

Менее тривиальными дефектами являются «ленивый» класс [FOWL, стр. 94] и большой класс (god-класс). Если класс не содержит в себе ничего, кроме открытых полей (или поля закрыты и к ним предоставлены методы доступа) - то скорее всего, в системе есть класс, который активно использует «ленивый» класс просто как структуру данных – god-класс. Следует проанализировать использование такого класса, и, возможно, перенести в него часть функциональности, содержащейся в god-классе.

Из этих примеров видно, что автоматическое нахождение дефекта проектирования в программном коде системы может быть весьма нетривиальной задачей. Если в случае с длинным телом метода достаточно просто посчитать количество строк, то для обнаружения ситуации «ленивого» класса нужно провести сложный анализ зависимостей класса, подсчета фактов использования класса и его членов, структуры класса и т.д. Ниже будут описаны подходы к анализу программного кода и поиску в нем дефектов.

Непосредственный анализ программного кода

Наиболее очевидным методом поиска дефектов является разбор и анализ программного кода с помощью специализированных парсеров. В процессе разбора кода извлекаются необходимые для анализа и выводов данные. Это дает создателю аналитического инструмента поистине безграничные возможности по настройке: степень детализации разбора, категории извлекаемых фактов и т.п. – все под его контролем.

Одним из аналитических средств, основанных на этом подходе, является jCOSMO - инструмент для поиска дефектов в программах на Java. Архитектура этого средства и особенности реализации подхода описаны в работе E. Emden и L. Moonen [EmdenMoonen]. Для каждого дефекта выводится ряд *аспектов* или особенностей (например, «в методе m встречается оператор switch») и в процессе разбора кода найденные аспекты помещаются в репозитарий для дальнейшего анализа. В качестве примеров рассматривается поиск часто встречающихся операторов `instanceof` и операций по явному приведению типов при работе с коллекциями объектов. Интересной особенностью инструмента является визуализация результатов поисков дефектов, которые представляются в виде графа, вершины которого представляют классы. Индикатором количества найденных в классе дефектов служит интенсивность цветовой окраски вершины графа (чем больше дефектов – тем ярче цвет). К сожалению, инструмент разработан как отдельное приложение и не является встраиваемым ни в одну из существующих популярных сред разработки (IDE).

Надо отметить, что поиск дефектов, основанный на простом разборе кода и собирании нужных фактов помимо неоспоримых достоинств, указанных выше, имеет существенный недостаток. Эвристические правила, определяющие дефект кода, жестко заданы в программном коде аналитического инструмента. Это весьма затрудняет их прочтение пользователем. При добавлении нового дефекта, фактически, потребуется доработка самого инструмента анализа. В следующих разделах будут рассмотрены подходы, позволяющие расширять возможности аналитического инструмента без существенных затрат на его доработку.

Использование объектно-ориентированных метрик

Объектно-ориентированные метрики широко используются для анализа качества проектирования информационных систем. Выступая в роли количественных критериев качества проектирования, метрики способны указать на источник возможных проблем и характер ошибки.

R. Martin в своих работах [Martin1, Martin2] определяет ряд метрик, вычислив которые для конкретной объектно-ориентированной системы, можно получить информацию о степени связанности и сцепленности компонентов системы, состоянии иерархий наследования классов и др., а также возможно делать некие выводы относительно дальнейшего развития системы. Определяя метрику, R.Martin перечисляет возможные проблемы, индикатором которых эта метрика может служить. Например, метрика CBO:

CBO (Coupling Between Objects) — связанность между объектами. Данная метрика измеряет степень сцепления (или зависимости) отдельно взятого класса от других классов. Подсчитываются классы, на который ссылается исходный класс при объявлении атрибутов, параметров, исключений в методах и локальных переменных. Примитивные типы и родительские классы не учитываются.

Избыточная связанность классов негативно сказывается на проекте системы и уменьшает возможность повторного использования класса. Чем больше независим класс от других, тем легче его использовать в других приложениях. Для увеличения модульности, внутренние связи класса с другими классами должны быть сведены к минимуму. Чем больше число таких связей, тем труднее вносить изменения в проект
системы и тем труднее таким классом управлять. Большое значение метрики сцепления для класса указывает также и на то, что данный класс следует подвергнуть особенно тщательному тестированию. В качестве еще одного примера рассмотрим метрику PMIS:

PMIS (Proportion of Methods Inhertied by Subclass) — пропорция методов, унаследованных дочерним классом. Данная метрика вычисляется как отношение количества унаследованных методов к количеству всех методов класса.

С помощью этой метрики, мы получаем степень специализации данного класса по отношению к своим родительским классам. Высокое значение метрики свидетельствует о том, что дочерние классы несут в себе мало дополнительной функциональности и иерархия наследования неоправданно раздута. В таких случаях имеет смысл объединить какие-то родительские и дочерние классы.

Рекомендации, приводимые в определениях метрик достаточно общие и сами по себе значения метрик мало что могут сказать. Для получения более конкретных указаний необходима интерпретация полученных значений метрик. При решении задачи поиска конкретных недостатков кода предполагается целесообразным комбинировать

различные наборы метрик а также добавлять собственные метрики, ориентируясь на порядок описания, заданный в работе [Martin1].

Frank Simon, Frank Steinbrückner и Claus Lewerentz в работе [MetrBsdRef] проводят поиск и трехмерную визуализацию мест программной системы для применения методов рефакторинга Extract/Inline Class, Move Method/Field. В статье вводится метрика для измерения расстояния между членами классов следующим образом:

$$dist_{A\cup B}(x,y) = 1 - \frac{|p(x) \cap p(y)|}{|p(x) \cup p(y)|}$$

где x и y - члены класса A или B (поля или методы) и $p(z) = \{p_i \subset A \cup B \mid z \to p_i\}$.

Далее анализируются расстояния между членами классов. Например, если один из методов класса чересчур близок к членам другого класса – это явный кандидат на перемещение в другой класс. Такой метод слишком интересуется устройством чужого класса, не используя класс, в котором содержится. Таким образом, выявляется дефект завистливых функций. Аналогичнымобразом в статье описывается поиск завистливых
классов.

Radu Marinescu в работе [DetDesFlaw] применяет весьма перспективный метод поиска недостатков кода: для каждого из рассматриваемых дефектов определяется комбинация метрик (автор определяет некоторые собственные метрики), на которые влияет существование этого недостатка кода. Для программной системы собирается статистика значений метрик и проводится количественный анализ. Для каждой метрики
определяется диапазон допустимых (или «безопасных») значений.Если все метрики в заданном наборе не попадают в этот диапазон, то в данном месте программного кода подозревается наличие конкретного недостатка кода, определяемого набором метрик. Полученные подозрительные места в коде выдаются эксперту, который должен вручную проанализировать код и подтвердить или опровергнуть наличие недостатка. Этот
подход иллюстрируется на примере поиска «классов данных» (или «ленивых» классов) и god-классов. Для поиска классов данных автор использует следующие метрики:

1. ***WOC (Weight of Class)*** — вес класса. Вычисляется как отношение количества собственных методов класса, не являющихся методами доступа к полю класса (get/set – методы) к общему количеству методов в классе.

2. ***NOPA (Number of Public Attributes)*** — количество открытых атрибутов. Вычисляется как количество собственных (не унаследованных) открытых (public) атрибутов класса.
3. ***NOAM (Number of Accessor Methods)*** — количество методов доступа к полям класса.

Получив значения этих метрик для каждого класса в системе, рассматриваются классы со значением метрики WOC в интервале [0, 1/3], т.е. классы, не содержащие в себе значительного собственного функционала. Для таких классов проводится анализ значений метрик NOPA и NOAM. В качестве «подозреваемых» на наличие недостатка выбираются классы, у которых NOPA > 5 и/или NOAM > 3. Таким образом, из всех классов в системе отфильтровываются те, которые не привносят новой функциональности и содержат в себе много данных, к которым открыт общий доступ. Эта ситуация в точности соответствует недостатку кода – «ленивым» классам. Аналогичным образом в работе [DetDesFlaw] рассматривается поиск god-классов.

Схожий подход использован при создании инструментального средства ProDeOOS (Problem Detection in OO Systems) [PRODEOOS], который доступен для работы через Internet. Пользователь может загрузить на сайт программы исходные тексты программной системы (на C++ или Java) и провести поиск дефектов в коде. В настоящий момент ProDeOOS поддерживает поиск следующих дефектов:

- data classes
- god classes
- shotgun surgery
- ISP violation
- refused bedquest
- feature envy
- god method
- misplaced class
- помимо этого, поддерживается поиск мест для применения некоторых шаблонов проектирования (design patterns)

Таким образом, используя объектно-ориентированные метрики возможно проводить поиск определенных дефектов кода. Но использование метрик в этих целях требует решения двух непростых задач: определения необходимого набора метрик для конкретного дефекта кода и интерпретации собранной статистики значений метрик. Добавление нового дефекта в систему поиска может оказаться сложной задачей: возможно, потребуется определить новые метрики, задать новые правила интерпретации.

К тому же, запись правил поиска дефекта кода как набора метрик и правил интерпретации довольно сложна и понять в чем, собственно, заключается дефект из нее непросто. Желательно, чтобы определение дефекта было приведено в удобочитаемой форме.

Поиск многих дефектов кода может требовать выполнения некого структурного анализа класса, иерархии наследования, поведения класса и т.п. Примером такого дефекта является родительский класс, включающий в себя некие сведения о своих потомках [Ciupke]. Искать подобные дефекты с помощью метрик весьма проблематично. Эти проблемы (сложность записи и возможность структурного анализа) решает подход, основанный на логическом мета-программировании, описанный в следующем разделе.

Использование логического мета-программирования (ЛМП)

Отправной точкой анализа системы с использованием ЛМП является получение информации о внутреннем устройстве системы (её дизайне). Вся эта информация организуется в некую структуру, называемою метамоделью системы. Метамодель системы может включать в себя очень многое: информацию о строении классов (списки атрибутов, методов, спецификаторы доступа к членам класса), описание взаимоотношений между классами, информация о взаимосвязи компонент системы и т.п. Метамодель может быть представлена как типизированный граф или задана как множества и отношения между ними, а также быть представленной как набор утверждений [Ciupke].

Основываясь на информации, содержащейся в метамодели, можно производить поиск определенных шаблонов или структур, выражающих дефекты кода. В данном случае выполнение структурного анализа не является проблемой – вся необходимая информация есть в метамодели. Если метамодель представлена как набор утверждений, то эксперт может сформулировать логические запросы к ней. Запросы могут выражать какой-то принцип проектирования систем или структурные зависимости, характеризующие определенные дефекты кода. Полученные результаты запроса должны быть проанализированы экспертом на предмет установления существования дефекта или нарушения правила проектирования.

В работе [Ciupke] O.Ciupke описывает анализ объектно-ориентированных систем с помощью логических запросов к метамодели системы. Также приводится подробное описание архитектуры инструментального средства GOOSE [GOOSE], позволяющего

проводить reverse engeneering системы и выполнять запросы на языке Prolog к полученной метамодели. GOOSE предоставляет возможность воспользоваться как готовыми запросами, выражающими принципы проектирования систем (например, «родительский класс не должен ссылаться на дочерние классы» или «корневой класс в иерархии должен быть абстрактным»), так и составить свой запрос. К сожалению, GOOSE не является встраиваемым ни в одно из современных популярных средств разработки, что сказывается на возможностях его промышленного использования негативным образом.

В работе [TorweMens] Tom Tourwé и Tom Mens описывают аналитический инструмент, построенный на использовании языка ЛМП SOUL, позволяющий проводить поиск дефектов в коде и предлагающий для каждого дефекта набор методов рефакторинга, которые его устраняют. Язык SOUL (Smalltalk Open Unification Language) весьма схож с языком Prolog. Он был разработан Roel Wuyts в качестве средства синхронизации дизайна системы, выраженного предикатами языка, и ее программного кода [SOUL]. Авторы используют предикаты на языке SOUL для задания правил, определяющих структуры программного кода, соответствующие определенным дефектам. Например, для дефекта кода «избыточный параметр» [FOWL, стр. 280] предлагается следующее правило:

```
obsoleteParameter(?class,    ?selector,?parameter)
:-
[1] classImplements(?class,?selector),
[2] parameterOf(?class,?selector,?parameter),
[3]
forall(subclassImplements(?class,?selector,?subcl
ass),
[4]
not(selectorUsesParameter(?subclass,?selector,?pa
rameter))
```

Этим правилом проверяется, используется ли параметр **?parameter** метода **?selector** в классе **?class** самим методом или перекрывающими его методами в потомках класса, содержащего исходный метод. Соответственно, все методы, удовлетворяющие
этому правилу будут выданы эксперту в качестве подозреваемых на наличие дефекта.

Для каждого дефекта определяется правило (опять же на языке SOUL) определяющее, какие методы рефакторинга и с какими параметрами можно применить к данному дефекту. В случае с

«избыточным параметром» это правило будет выглядеть следующим образом:

```
proposeRefactoring(?class,removeParameter,
<?class,?selector,?parameter>) :-
obsoleteParameter(?class,?selector,?parameter)
```

Данный подход авторы воплотили во инструментальном средстве, встраиваемом в среду разработки на Smalltalk – Visual Works IDE. Это средство является расширением для Refactoring Browser [Roberts] – стандартного инструмента этой среды, выполняющего поддержку проведения рефакторинга. Это пример идеальной интеграции средств анализа и рефакторинга – оба средства взаимосвязаны (средство анализа предлагает варианты
методов рефакторинга, которые могут быть выполнены с помощью Refactoring Browser), легко доступны и работа с ними происходит в рамках привычной среды разработки.

Комбинированный подход

Рассмотрев дефекты кода, приводимые в [FOWL], можно отметить, что при описании дефекта зачастую указываются структурные особенности и некие количественные характеристики рассматриваемого кода. Рассмотрим, например, дефект кода «Классы данных» [FOWL, стр. 98]: класс является классом данных, если он содержит в себе поля, методы доступа к ним и не содержит значимого функционала. При этом не
исключается возможность наличия 1-2 методов отладочного характера, например, метода, возвращающего строковое представление экземпляра класса (toString() в Java). В данном случае, структурная информация - это информация о методах, содержащихся в классе (они должны являться методами доступа к полям класса или не нести в себе значимого функционала). Информация о количестве таких методов доступа и соотношении с количеством «функциональных» методов - это уже количественные характеристики. Само же разграничение методов на «функциональные» и «не функциональные» может быть проведено также с помощью метрического анализа этих методов (например, их размера и цикломатической сложности [Martin1]).

Другим примером дефекта, в описании которого используются структурные и метрические характерискики кода, является дефект «Операторы типа switch»: если в коде класса слишком часто встречаются блоки следующего вида:

```
switch (someClassVariable) {
   case A: doSomethingA();
     break;
   case B: doSomethingB();
     break;
   default: doDefault();
}
```

или

```
if (someClassVariable == A) {
   doSomethingA();
} else if (someClassVariable == B) {
   doSomethingB();
} else {
   doDefault();
}
```

то скорее всего, такие блоки следует заменить полиморфизмом (например, вынести повторяющиеся блоки в отдельные виртуальные методы и перекрывать их в подклассах). Для данного дефекта описываются как структурные характеристики – наличие в коде определенных операторов, считывающих значение переменной класса, так и количественные – таких операторов должно быть несколько (например, больше трех).

Таким образом, представляется целесообразным использовать одновременно как структурную информацию, так и значения метрик при поиске недостатков кода. Необходимо также отметить, что при поиске дефектов нередко требуется вычислять значение весьма нетривиальных метрик (например, количество операторов switch, считывающих значение конкретной переменной класса).

Для такого поиска дефектов предлагается использовать расширенные средства логического мета-программирования. Добавление в средство ЛМП предикатов, выполняющих арифметические действия, операции сравнения и вычисляющих мощность множества истинности другого предиката, способно существенно расширить возможности инструмента и предоставить способы для вычисления значений искомых нетривиальных метрик. Используя такие предикаты, можно, например, составить следующее правило поиска дефекта кода «Классы данных»:

```
DataClass(?C) :- class(?C),
```

```
COUNTALL(getter(?C,      ?getMeth),      ?getMeth,
?getterQty),
COUNTALL(dataField(?C,   ?dataField),   ?dataField,
?dataFieldQty),
sum(?getterQty,      ?dataFieldQty,      ?dataQty),
greater(?dataQty, 4),
COUNTALL(ownMethod(?C,     ?ownMeth),     ?ownMeth,
?ownMethQty),
NOT(greater(?ownMethQty, 2)).
```

В данном правиле используются следующие предикаты:

- COUNTALL(?query, ?variable, ?result) — возвращает в переменной **?result** количество вхождений значения **?variable** в результаты запроса **?query**
- sum(?source, ?augend, ?result)} — возвращает в переменной **?result** сумму значений **?source** и **?augend**
- greater(?comp1, ?comp2) — истинно, если **?comp1** больше **?comp2**

Кроме того, используются дополнительные предикаты **getter**(?Class, ?Method), определяющий является ли метод класса методом доступа к полю класса, и **dataField**(?Class, ?Field), определяющий является ли поле класса полем данных.

Таким образом, расширив средство ЛМП вышеописанными предикатами, мы получаем возможность поиска определенных структурных и количественных характеристик кода, описывающих искомый дефект. При этом описание дефекта сохраняет высокую степень читаемости и предоставляет возможности для настройки поиска (например, путем изменения пороговых значений для метрик).

Заключение

Проблема автоматизированного поиска дефектов программного кода объектно-ориентированных систем весьма актуальна для промышленной разработки программного обеспечения. Динамически изменяющиеся внешние условия, постоянная смена состава команд разработчиков, нечеткость и изменчивость требований заказчиков - эти условия влекут за собой жесткие требования к качеству кода разрабатываемых систем. В таких обстоятельствах автоматизированный инструмент контроля качества кода может оказать немалое подспорье. Такие инструменты должны удовлетворять следующим требованиям:

- **Оперативность** – время анализа кода может существенно варьироваться в зависимости от исходных условий (размера анализируемого кода, критериев поиска и пр.), но в любом случае, должно укладываться в разумные рамки. Чем дольше будет время анализа кода, тем реже этот анализ будет проводиться.
- **Расширяемость** – в процессе работы над системой могут быть получены новые эвристические правила проектирования и кодирования. Аналитический инструмент должен поддерживать возможность создавать пользовательские описания дефектов (и выполнять поиск по ним).
- **Возможность интеграции** – значительная часть программистов работает в интегрированных средах разработки (IDE). Это удобные средства, включающие в себя редактор кода, отладчик, управление файлами и пр., т.е. необходимые при разработке компоненты, которые постоянно используются в процессе написания программного кода системы. Анализ кода должен проводиться регулярно, и вынесение аналитического инструмента в отдельное приложение снижает его шансы на частое использование. С учетом того, что многие популярные среды разработки поддерживают механизм подключаемых модулей (plugins), представляется целесообразным реализовывать аналитический инструмент в виде такого модуля.

Литература

1. [FOWL] Martin Fowler «Refactoring. Improving Design Of Existing Code»
2. [XP] Kent Beck «Extreme Programming Explained»
3. [Opdyke] William Opdyke «Refactoring Object-Oriented Frameworks»
4. [Roberts] Donald Roberts «Pratical Anlaysis For Refactoring»
5. [EmdenMoonen] Eva van Emden, Leon Moonen «Java Quality Assurance by Detecting Code Smells»
6. [Martin1] Robert Martin «A Metrics Suite For Object-Oriented Design»
7. [Martin2] Robert Martin «OO Design Quality Metrics: An Analysis Of Dependencies»
8. [MetrBsdRef] Frank Simon, Frank Steinbrückner, Claus Lewerentz «Metrics Based Refactoring»
9. [DetDesFlaw] Radu Marinescu «Detecting Design Flaws via Metrics in Object-Oriented Systems»

10. [PRODEOOS] ProDeOOS Project Home Page http://193.226.12.248/prodeoos/index.php
11. [Ciupke] Oliver Ciupke «Automating Detection of Design Problems in Object-Oriented Reengineering»
12. [GOOSE] GOOSE Project Home Page http://esche.fzi.de/PROSTextern/software/goose/index.html
13. [TorweMens] Tom Torwé, Tom Mens «Indentifying Refactoring Opportunities Using Logic Meta Programming»
14. [SOUL] Roel Wuyts «A Logic Meta-Programming Approach to Support the Co-Evolution of Object-Oriented Design and Implementation»

Леонов М.В., Мошкин К.Б.

Программный инструментарий для автоматизированного описания и идентификации сложных объектов на основе XML-технологии.*

Введение

Задачу автоматизированного описания сложных объектов сформулируем следующим образом. Пусть имеется значительное количество родственных объектов, которые нужно описать на естественном языке в терминах определенного набора признаков и их значений. Ясно, что названия признаков и значения их будут повторяться для некоторых, а может и для многих объектов. При этом как множество объектов, так и множество их признаков обычно имеют иерархическую структуру. Требуется облегчить процесс формирования описаний, разработав, во-первых, адекватную для обработки структуру данных для хранения, а во-вторых, удобный интерфейс для пользователя, редактирующего такие описания. Эта задача весьма актуальна для зоологии и ботаники, а также для других описательных наук, но не только там: вспомним биографические справки с фразами "морально устойчив", "характер нордический" и т.д. Когда речь идет о сотнях и тысячах объектов, разумно процесс составления таких описаний автоматизировать.

Вторая задача, связанная с первой – автоматизированная идентификация описанных объектов по заданным значениям их признаков. Она решается с помощью программ обработки описаний, созданных при решении первой задачи; в частности, путем генерации дихотомических определителей, которые являются обязательной составляющей таксономических монографий.

Одна из удачных для своего времени попыток такой автоматизации была предпринята в 70-х годах прошлого века программистами, которые сотрудничали с зоологами. Система морфологических описаний таксонов была формализована и реализована в виде пакета программ (на языке ФОРТРАН) под названием DELTA (Description Language for Taxonomy) применена в зоологии, а затем и в ботанике [6]. Во время разработки системы DELTA реляционные базы данных еще не получили распространения. Впоследствии попытки использовать технологию баз данных к задаче

* Работа выполнена в соответствии с задачами проекта, поддержанного грантом **РФФИ 04-07-90303**

автоматизированного описания в ботанике и зоологии большого успеха не имели. Это можно объяснить, во-первых, тем, что обычно и множество объекты, и множество их признаков имеют иерархическую структуру, а, как известно, иерархические структуры не очень удобно вписываются в реляционную модель данных БД. Во-вторых, процесс научной классификации по своей природе – итеративный и поэтому у специалиста предметной области часто меняется не только число признаков, но и глубина иерархии. В рамках реляционной модели удачного решения для задач с такими условиями так и не было предложено. В настоящее время существует некоторое количество удачных приемов для преодоления возникающих трудностей, но естественными такие модели назвать трудно [4].

Ситуация, на наш взгляд, изменилась с появлением XML-технологии. В данной работе мы представляем свой подход и программу для решения вынесенных в заголовок задач применительно для таксономической ботаники.

• 1. Модель данных и выбор инструментальной среды .

Для описания модели данных, а также для организации среды хранения данных использовано подмножество разработанного нами для описания ботанических монографий языка BTML (Botanical Taxonomic Markup Language), основанного на языке разметки XML.

Выбор языка XML для описания модели и в качестве средства хранения данных при решении указанных задач обусловлен следующими соображениями.

1. Реляционная модель не является оптимальной для использования в задачах с естественными иерархическими структурами, то есть в частности, и в данном случае. А технология XML предлагает готовое решение описания иерархических структур.
2. Модель данных в XML-представлении отличается понятностью, что существенно для наших коллег-биологов, текст на языке XML несравненно более прозрачно отражает структуру таксономической монографии, чем совокупность ER--диаграмм или реляционных таблиц.
3. Принадлежность языка XML к Интернет-технологиям обеспечивает простоту организации Интернет-доступа к данным в этом формате.

Есть и другие аргументы в пользу такого выбора [3]. Отметим еще один, нейтрализующий упреки в недостаточной скорости обработки XML-документов: «сегодня удобство проектирования и

эксплуатации баз данных экономически важнее, чем скорость обработки на компьютере» [1].

В нашем случае модель данных можно представить в виде трех деревьев: дерева таксонов, дерева признаков и дерева назначений, создаваемое пользователем на основе двух первых. В актуальной версии дерево таксонов имеет три уровня: семейство, род, вид; дерево признаков также имеет три уровня – группа признаков, признак, значение.

На рис 1 представлен фрагмент XML-документа для дерева таксонов на примере рода Heracleum.

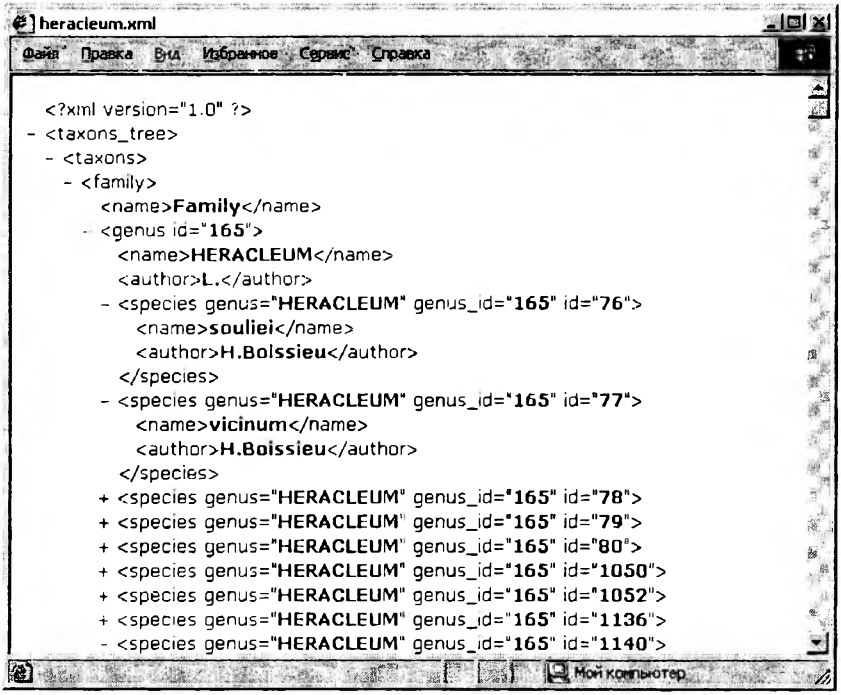


Рис 1. Фрагмент документа с деревом таксонов.

На рис. 2. аналогичным образом представлен фрагмент дерева признаков, а рис. 3. изображает дерево назначений, то есть дерево с перекрестными ссылками, на основе которого можно генерировать текстовое описание элементов дерева таксонов.

Таким образом, наша задача сводится к разработке программы, которая позволяет удобным образом манипулировать (то есть добавлять новые элементы, уничтожать и редактировать имеющиеся элементы) с деревом таксонов, с деревом признаков, а также для заданных таксонов выбирать нужные признаки и указывать их значения.

В качестве среды разработки такой программы (названной MDELTA) была выбрана платформа .NET Framework с ее компонентами CLR (common language runtime, общеязыковая исполняющая среда) и FCL (библиотекой классов .NET Framework) [5]. FCL предоставляет объектно-ориентированный API, к которому обращаются приложения. Благодаря CLR, приложения, разработанные на этой платформе, могут выполняться под управлением любой ОС. Приложения компилируются в промежуточный код, а в момент исполнения преобразуются в машинный. При этом происходит проверка преобразования типов, что увеличивает надежность программы.

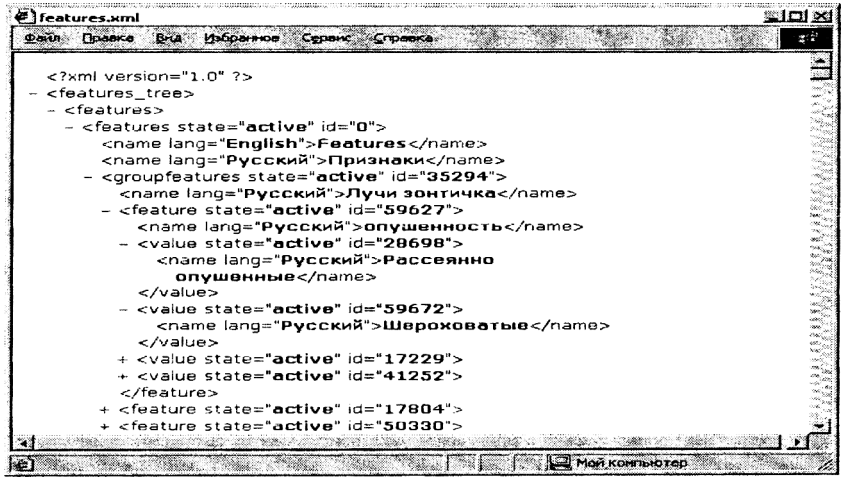


Рис 2. Фрагмент документа с деревом признаков.

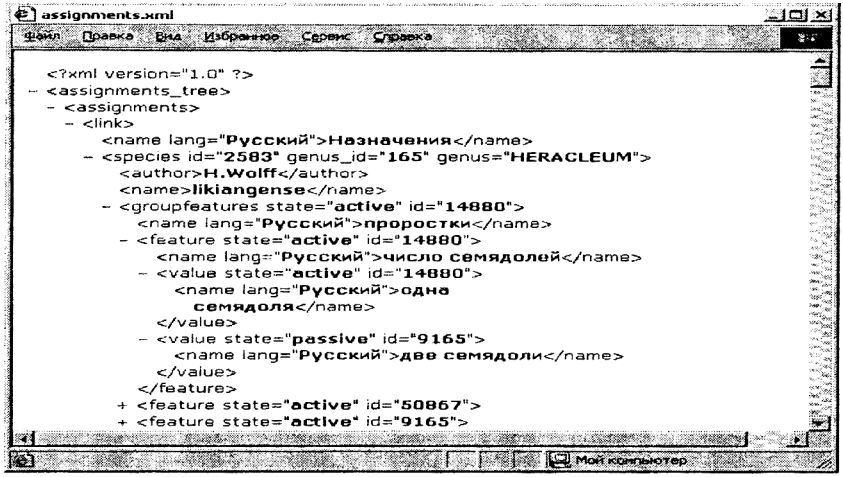


Рис 3. Фрагмент документа с деревом назначений.

Языком реализации был выбран язык C#, весьма лаконичный, но обладающий всеми возможностями современного языка программирования. В отличие от широко распространенного C++, C# более строгие правила написания кода, что ведет к снижению числа ошибок и уменьшению времени написания и отладки кода программы.

2. Функции программы и ее интерфейс.

На рис. 4. изображена главная панель программы с тремя окнами, каждое из которых содержит пользовательское представление соответствующих деревьев.

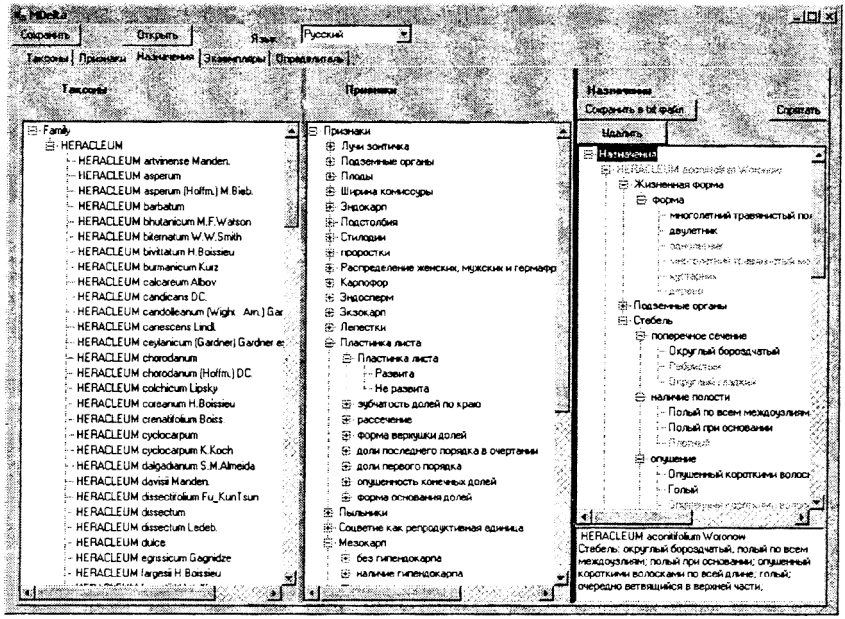


Рис.4. Интерфейс программы в момент генерации описаний.

При помощи закладок «Таксоны», «Признаки» и «Назначения» активизируются отдельные окна, в которых можно выполнять операции редактирования соответствующих деревьев. Во второй верхней строчке кнопки «Открыть» и «Сохранить» обеспечивают загрузку и сохранение данных в виде XML-документов. Кнопка «Сохранить в txt-файл» позволяет сохранить результат работы в виде текста, привычного для специалистов предметной области. Предусмотрена возможность хранить названия и значения признаков на нескольких языках. Переключение языка делается через список выбора во второй строке (см. рис. 1).

Закладка «Определитель» открывает режим фильтрации объектов (таксонов) по отобранным признакам и их значениям, то есть обеспечивает функцию политомического определителя. Это свойство программы дает пользователю использовать ее в качестве информационно-аналитической системы.

Тройки XML-документов «таксоны, признаки, назначения» можно считать XML-описаниями таксонов соответствующей предметной области. Таксон – это, вообще говоря, абстрактный класс объектов, встречающихся в природе. Иногда требуется хранить данные по конкретным экземплярам таксонов, например, представленным в некоторой коллекции. Для этого в программе предусмотрен режим

работы со структурой, где кроме уровней «семейство, род, вид» дополнительно присутствует уровень «экземпляр». Этот режим открывается закладкой «Экземпляры».

Заключение

Программа MDELTA, разработанная для нужд наших коллег ботаников из Ботанического сада МГУ[2], может служить примером системы управления XML-описаний некоторого множества объектов, к которой предъявляются требования, сформулированные во введении. Программа имеет функцию автоматической генерации описаний объектов на естественном языке, а также функцию определителя объекта по значениям признаков. В следующих версиях системы предполагается управлять хранением и предъявлением изображений, характеризующих признаки и их состояния. Опыт разработки программы и ее опытной эксплуатации, по нашему мнению, говорит о целесообразности использования языка XML не только в качестве формата для обмена данными между приложениями, но и в качестве структуры для хранения и обработки данных относительно небольшого объема в задачах указанного класса.

Литература.

1. Веселов В., Долженков А. Опыт построения XML-СУБД.// Открытые системы, № 6, 2002.
2. Леонов М.В. Комплекс программ для автоматизации исследований в таксономической ботанике. / Сб. "Программные системы и инструменты" №2, - М., 2001, стр.168-172.
3. Леонов М.В., Мошкин К.Б., Леонов В.М. XML как средство модернизации унаследованных баз данных. / Сб. "Программные системы и инструменты" №3, - М., 2002, стр.28-32.
4. Лобанов А.Л. и др. Современное состояние концепции ZOOCOD/ Материалы Международного симпозиума "Информационные системы по биоразнообразию видов и экосистем", Зоологический институт РАН, Санкт-Петербург, 2003 г., стр. 8-9.
5. Просиз Джеф. "Программирование для Microsoft .NET" Пер. с англ. - М.: Изд-во "Русская Редакция", 2003. - 704 стр.
6. Dallwitz, M. J. A general system for coding taxonomic descriptions. //Taxon № 29, 1980, pp.41–46.

Глазов А.Э., Леонов М.В., Новиков М.Д.

Элементы дистанционного обучения и контроля знаний на вечернем отделении факультета ВМиК.

Введение

Особенностью обучения на вечернем отделении факультета (разумеется, не только нашего) являются разнообразные объективные трудности, следствием которых становятся пропуски занятий, по уважительной причине в частности. Кроме того, распространенность личных компьютеров, почти повсеместная возможность доступа в Интернет и использования электронной почты позволяют преподавателю дать студенту возможность выполнять при желании практические занятия на своем компьютере, и встречаться с преподавателем практикума только для консультаций и сдачи контрольных заданий. Современные Интернет-технологии позволяют автоматизировать этот процесс с относительно небольшими затратами. Опыт такой автоматизации и послужил основой нашего доклада.

1. Постановка и актуальность задачи.

Известно, что в настоящее время по инициативе «снизу», то есть самими студентами создаются Web-сайты для обмена информацией, архивами учебных материалов, общения между собой. По нашему мнению, естественно и преподавателям участвовать в этом процессе. У нас уже был опыт создания и сопровождения сайта спецсеминара «Интернет технологии в научных исследованиях», поэтому на его основе было решено разработать сайт с авторизованным доступом для студентов вечернего отделения. Этот сайт должен содержать информационные разделы (с архивами электронных документов, с доской объявлений, библиографическими ссылками – книжными и на Интернет-источники), форум для общения студентов между собой и преподавателями, систему электронной рассылки, а также систему дистанционного тестирования. Эта систему предполагалось использовать в основном для проверки знаний по отдельным темам практикума. Заметим, что мы не преследовали цели создания тестов по научно-обоснованным методикам тестирования с их критериями надежности и валидности. Нам было важно иметь удобную оболочку, которую можно рекомендовать студентам в качестве тренажера по языку Паскаль, а также использовать и как средство

проведения автоматизированного анкетирования, а также проведения письменных экзаменов в компьютерном классе.

Кроме того, несмотря на дискуссионность широкого использования автоматизированного тестирования, нельзя не признать, что эта форма является неплохим дополнением к традиционным формам контроля усвоения знаний. Можно вспомнить, например, высказывания А.П. Ершова о том, что «...по многим показателям ЭВМ является гораздо более удобным ...[по сравнению с преподавателем] контролером знаний» [1, стр.39]. Опыт показывает, что элемент игры, вносимый тестированием, полезен не только детям, но и взрослым учащимся.

2. Архитектура и реализация сайта

Для того, чтобы сайт одновременно был пригоден для обслуживания нескольких групп, работающих по разным программам (практикум по языку Паскаль, практикум по ОС UNIX, спецсеминар и т.д.) в модели данных сайта были выделены следующие сущности:

- Разделы сайта.
- Пользователи.
- Группы пользователей

Для групп пользователей предусмотрено отношение «права доступа к разделу». Объекты модели и отношения между ними могут быть представлены с помощью диаграммы, изображенной на рис. 1.

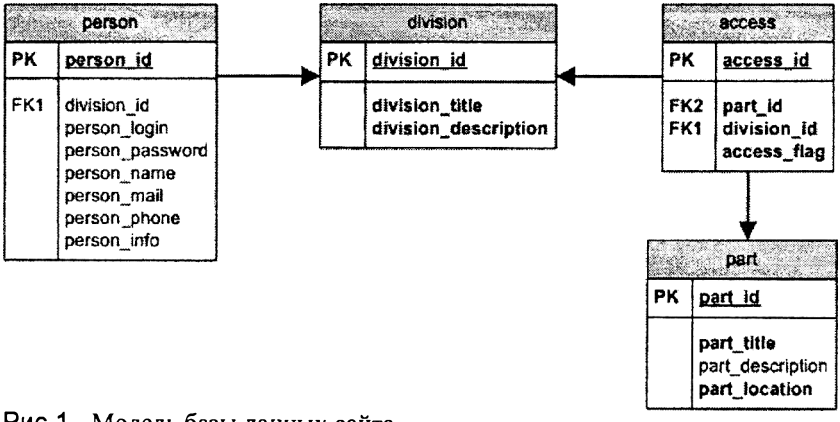


Рис.1. Модель базы данных сайта

При конструировании сайта было решено прибегнуть к подходу, суть которого состоит в следующем.

1. Все страницы создаются одной программой (скриптом), условно называемой движком сайта. Сайт может иметь несколько движков (для пользователя, для администратора).
2. Индивидуальная для каждой страницы информация представляется модулями, хранящимися в отдельных файлах.
3. Общая для группы страниц информация хранится в одном файле, называемом шаблоном. Сайт может иметь несколько шаблонов.
4. Шаблон представляет собой html-файл, в котором места индивидуальной для каждой страницы информации «почти пусты», точнее говоря снабжены специальными метками, с помощью которых движок будет определять, куда вставлять содержимое того или иного модуля.
5. Чтобы определить, какую страницу создавать в каждый конкретный момент, движок получает ее номер либо явно через запрос от браузера, либо берет по умолчанию из файла настроек.
6. Соответствие между номером страницы и требуемыми модулями и шаблоном определяется на основании настроек, хранящихся либо в файле, либо в базе данных.

Такой подход позволяет избавиться от избыточности информации, присущей конструированию сайтов из готовых html-страниц, когда некоторый код (заголовки, меню, и т.д.) многократно дублируется в каждой странице. В результате снижается требуемый объем ресурсов, а также упрощается редактирование общих частей.

Для сопровождения сайта существует режим администрирования, в котором можно регистрировать группы пользователей и редактировать их и их права доступа к разделам сайта, просматривать статистику посещения студентами, добавлять новые разделы и редактировать существующие.

В качестве инструментов для реализации были выбраны СУБД MySQL и скриптовый язык PHP. Сайт размещен на сервере факультета ВМиК и функционирует под управлением Web-сервера Apache.

Система тестирования ATS, которую можно считать универсальной оболочкой для тестирования, реализована с помощью тех же средств, что и сайт. Поэтому ее можно использовать и вне Интернета, на любой машине, где инсталлированы Web-сервер Apache, СУБД MySQL и скриптовый язык PHP.

3. Краткое описание модели данных в тестирующей системе ATS.

Для совокупности тестов по некоторой дисциплине мы использовали термин «курс» (по аналогии с системами автоматизированного обучения).

Вопросы в нашей тестирующей системе могут быть представлены в соответствии со следующей моделью:

- Система может содержать много курсов.
- Каждый курс может содержать много тем.
- Каждая тема может иметь несколько вариантов.
- В каждом варианте может быть несколько заданий
- Задания бывают двух типов: задания с готовыми ответами, предоставляющимися пользователю во время тестирования (верными могут быть несколько ответов), задания со свободным ответом, когда пользователь отвечает в произвольной форме.

Схема модели данных изображена на рис.2.

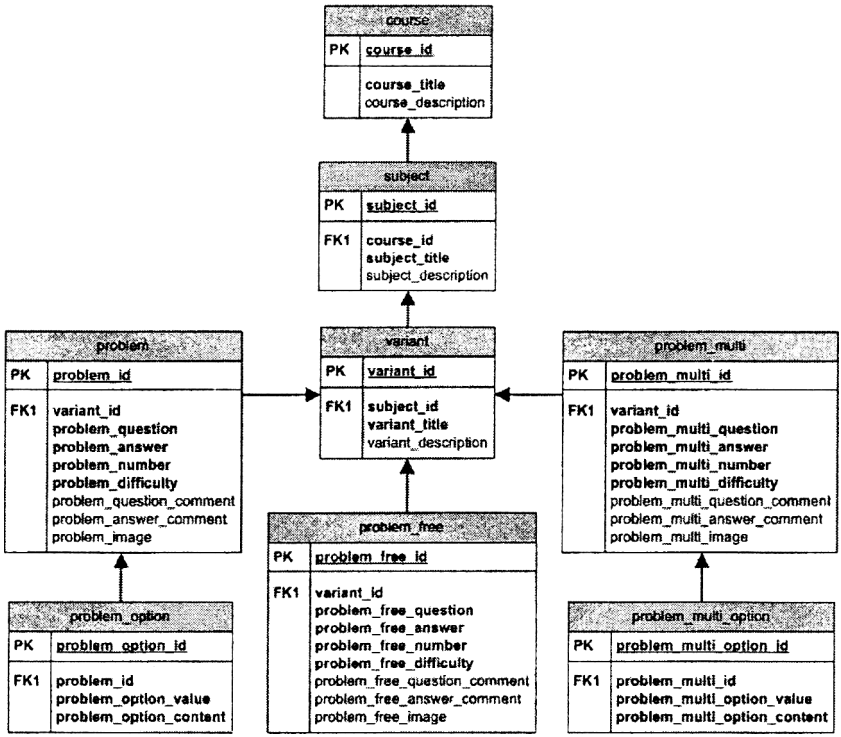


Рис.2. Модель данных в системе тестирования ATS

Таким образом, модель данных содержит следующие сущности:

- Курс (course)
- Тема (subject)
- Вариант (variant)
- Задание (problem)
- Вариант ответа (problem_option)
- Задание со свободным ответом (problem_free)
- задание со многими правильными ответами (problem_multi)
- вариант ответа к заданию со многими правильными ответами (problem_multi_option)

Вопрос может быть текстовым, а может содержать изображение, например, блок-схему алгоритма, и т.д.

4. Архитектура системы ATS.

Система ATS имеет два основных режима: режим администрирования и режим пользователя. При использовании системы

в Интернет этим режимам соответствуют два различных адреса. Оба входа являются авторизованными.
В режиме администрирования создаются группы пользователей, редактируется их состав, изменяются свойства группы. В этом же режиме осуществляется ввод и редактирование новых курсов, тем и вариантов, причем как в интерактивном режиме, так и в пакетном режиме. Для пакетного режима необходимо заранее подготовить файл с текстом в формате XML. Пакетный режим позволил использовать большой массив накопленных ранее тестов по языку Паскаль. Выбор языка XML для пакетного формата обусловлен тем, что XML «становится в последнее время доминирующим стандартом представления и обмена данными в Интернете» [2], а также и потому, что у нашего коллектива есть опыт разработки генераторов XML–документов на основе различных БД.

Для удобства использования системы при анкетировании абитуриентов предусмотрена возможность установки для группы пользователей свойства авторегистрации. Это означает, что перед сеансом тестирования пользователь должен сам себя зарегистрировать в системе, чтобы затем выполнить необходимый курс тестирования.

В пользовательском режиме существует возможность выбора курсов из некоторого множества, приписанного к группе, к которой принадлежит пользователь. Один и тот же тест пользователь не может пройти дважды. Время начала и окончания сеансов тестирования, как и вопросы и ответы на них, фиксируются в протоколе.

Статистика прохождения теста доступна только в режиме администрирования, ее можно просматривать даже в процессе тестирования, а также записать в файл в формате, удобном для вывода на принтер. Эту возможность естественно можно использовать для проведения письменного экзамена.

В актуальной версии системы не предусмотрен блок оценки результатов тестирования. Отчасти это связано с тем, что основную роль оценки результатов мы пока оставляем за преподавателем. Это оправдано еще и потому, что у нас многие тесты имеют так называемые открытые задания, то есть со свободным ответом. В этом смысле мы следуем не американской, а английской традиции тестирования. Если в американской школе тестирования контрольные измерительные тесты сводятся к тестам с вынужденным вариантом ответа, то в английских (в частности, в материалах Кембриджского экзаменационного синдиката) «доля открытых заданий традиционно выше заданий с вынужденным выбором ответа» [3].

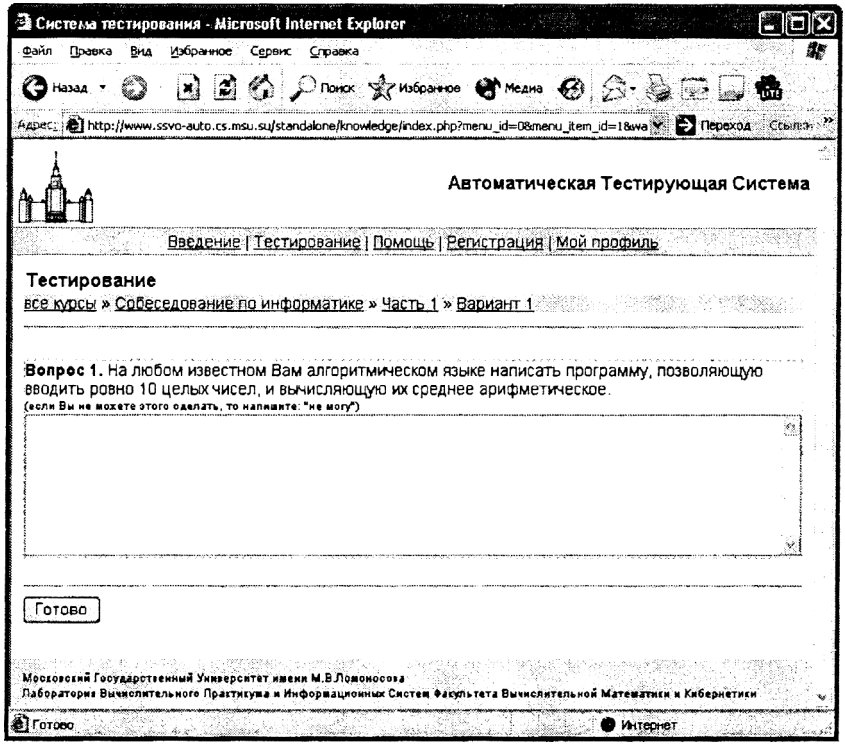


Рис. 3. Интерфейс системы тестирования в момент предъявления вопроса со свободным ответом.

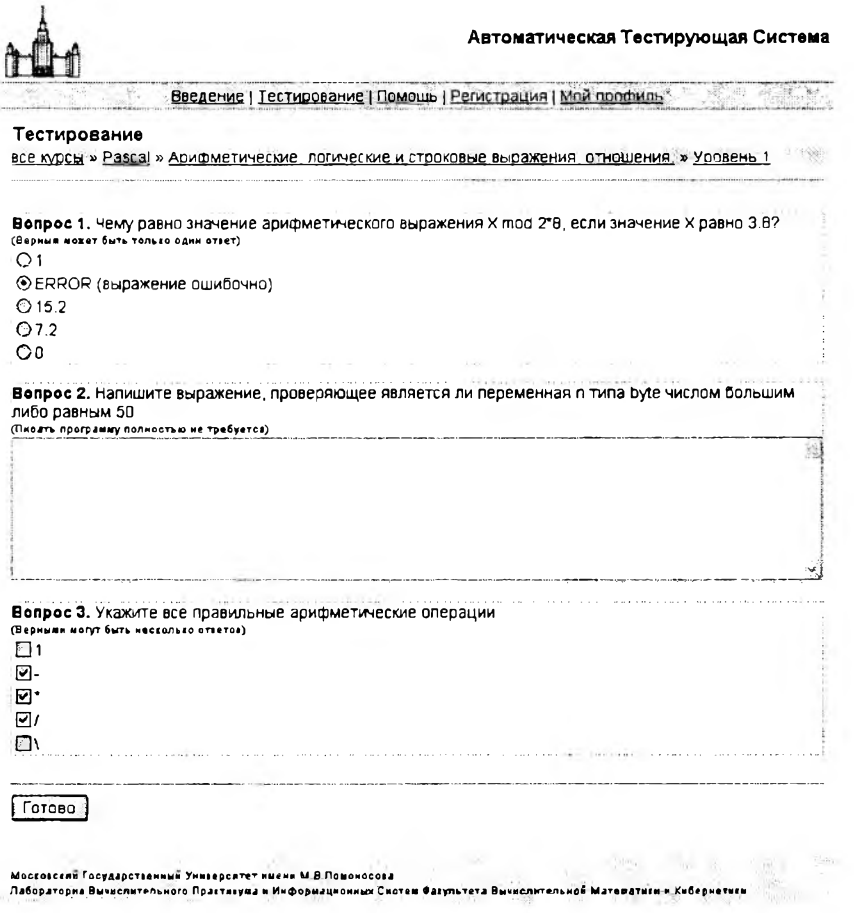


Рис. 4. Интерфейс системы в момент предъявления вопросов с тремя типами варианта ответа.

Заключение.

Описанные выше сайт и система дистанционного тестирования уже используется для собеседования по информатике и как тренажер по языку Паскаль. Благодаря наличию пакетного ввода тестов, а также разработанного на языке СУБД FoxPro редактора тестов, техническая сторона подготовки новых тестов особых трудностей не представляет. Кроме курса тестов по языку Паскаль и автоматизированного анкетирования с вопросами по информатике для поступающих на вечернее отделение факультета ВМиК, подготовлен курс по языку

оболочки bash операционной системы UNIX. Подготавливается небольшой экспериментальный курс с избранными вопросами по истории математического образования на основе пособия [4]. Кроме того, разрабатывается вариант системы тестирования, загружаемой для выполнения с CD- диска. Эти работы обусловлены договором о сотрудничестве факультета ВМиК с Елецким Государственным Университетом имени И.А. Бунина.

Системой тестирования можно пользоваться и через Интернет, а не только в учебном классе, что особенно удобно для указанной выше категории слушателей. Но в этом случае преподавателю приходится принимать во внимание дополнительные факторы, продумывать меры, повышающие вероятность самостоятельного выполнения тестов.

Опыт эксплуатации разработанных программных средств для дистанционного контроля и информационной поддержки учебного процесса показал востребованность указанного подхода. Особенно хочется отметить перспективные возможности совместного использования аналогичных систем для сотрудничества МГУ и провинциального ВУЗа.

Литература.

1. Ершов А.П. Избранные труды. – Новосибирск: ВО «Наука», 1994.-416 с.

2.Новак Л.Г., Кузнецов С.Д. Свойства схем данных XML, Труды Института Системного Программирования РАН, 2003..

3. Теплов П. Автоматизированная система оценки знаний. Открытые системы, декабрь 2002 г., стр. 70-72.

4. Кузовлев В. П., Перцев В. В., Саввина О. А. Контрольные тестовые задания по истории отечественного математического образования. Елец: ЕГУ им. И . А . Бунина, 2004. 44 с.

Полякова И.Н., Емашова О.А.

Автоматическое реферирование русскоязычных текстов. Редукция предложений

1. Введение

Одной из самых актуальных проблем современной компьютерной лингвистики является проблема автоматического реферирования текстовых документов. Активные исследования в этой области обусловлены большим количеством информации, хранящейся и обрабатываемой в цифровом виде, и значительными успехами в развитии всех областей компьютерной обработки данных. Зачастую информационные файлы содержат огромное количество побочной информации или ненужные подробности. В условиях хронического недостатка времени ознакомление с такими массивами информации нерационально, при условии, что нужно всего лишь получить общие представления по интересующему нас вопросу.

Разработка эффективных методов автоматического реферирования подразумевает сложный многоуровневый подход к проблеме. Исследования в области автоматического реферирования должны учитывать находки и методы экспертов в этой области, а так же использовать лингвистические знания о предмете текста и его структуре и, конечно, обращаться к различным сферам обработки естественного языка. Опыт разработок в области автоматического реферирования позволит оценить и сравнить уже предложенные методы, знания экспертов помогут добавить существенные факторы, благотворно влияющие на качество конечных рефератов. Внимание к проблеме дискурса позволит использовать множество связанных с ним феноменов, характеризующих различные тексты, что позволит эффективно использовать лингвистические знания о дискурсе, значительно вовлеченном в процесс реферирования. Знания о жанре и прочих подобных характеристиках текста также могут помочь в достижении наилучших результатов: зачастую жанр текста не просто стимулирует, но диктует способ или особенности метода реферирования, который должен быть применен в конкретном случае.

Кроме того, детальное изучение этих проблем позволит создать системы реферирования общего назначения, не зависящие от предметной области или жанровых особенностей текста. В свою очередь внимание, уделенное пользовательским запросам и привычкам, позволит сделать разрабатываемые системы простыми и удобными в использовании, создаст возможность настройки системы на конкретного пользователя или задачу.

2. Основные определения и понятия

По сути дела, обыкновенный пересказ является в некотором роде рефератом. Из чего можно сделать закономерный вывод, что проблема сокращения текста возникла вместе с появлением длинных текстов. Человек, прослушав информацию, делает вывод о том, какие предложения/темы оставить в пересказе, а какие – нет. Первые имеют непосредственное отношение к проблеме, освещенной в тексте, или несут в себе интересную информацию, новые методы, красивый подход и т.д. Вторые же содержат утомительные подробности или являются «лирическим отступлением» от темы. При этом важным фактором является сохранение связности текста и передача его смысла без искажений. Из приведенного выше рассуждения становится понятно, что реферирование – это процесс творческий, но часто довольно утомительный и скучный. Особенно если предмет, обсуждаемый в тексте, совсем не интересен читателю, и процедура реферирования только отнимает драгоценное время. Если человек, составляющий реферат, совсем не знаком с проблемой, которая освещена в документе, ему очень сложно решить, какие предложения и идеи следует оставить, а какие можно убрать. На данном этапе развития компьютерной лингвистики существует возможность переложить (хотя бы частично) решение этой задачи на компьютер.

Одним из важнейших этапов решения любой проблемы является правильная и точная ее формулировка. Конечно, можно сказать, что реферирование – это просто извлечение основных тем документа с одновременным уменьшением его объема. Такое определение дает нам общее функциональное описание процесса, однако едва ли поможет усовершенствовать уже существующие системы. Лучше представить общее действие «реферирование» как следующую совокупность: факторы, влияющие на процесс, и сам процесс. Таким образом, можно отдельно изучить каждый аспект, влияющий на конечный результат.

Итак, первое, что мы рассмотрим – это факторы, влияющие на процесс реферирования. Их можно разделить на три большие группы: исходные ограничения, ограничения цели и конечные ограничения. Исходные ограничения – это ограничения, обусловленные особенностями текста, который мы будем реферировать. Например, форма (скрытая и явная структурная организация текста), тема (предметная область, к которой относится документ) или «причастность» (текст реферируется как отдельный документ или как часть коллекции).

Целевые ограничения подразумевают особенности, обусловленные назначением генерируемого реферата. Например, чтобы сопровождать оригинал или же замещать его. В связи с целевыми ограничениями рефераты принято причислять к одной из следующих

категорий: *показательные (indicative)* и *информативные (informative)*. Первые используются как индикатор упомянутых в исходном документе тем. Таким образом, человек, ознакомившись с этим рефератом, может решить: стоит читать весь текст или нет. Информативные рефераты, в свою очередь, используются наравне с исходным документом, включают его основные описания, находки и заключения.

Конечные факторы включают формат вывода, стиль. Они частично определяются ограничениями ввода и цели, однако еще учитывают и более детальные выходные характеристики реферата.

Теперь рассмотрим сам процесс реферирования. Принято различать методы так называемого автоматического экстрагирования и автоматической генерации реферата.

• ***Автоматическое экстрагирование (квазиреферирование):*** Характерные параграфы, предложения или клаузы выбираются из исходного текста согласно некоторой оценке их важности для присутствия в реферате. Большинство ныне используемых методов квазиреферирования используют в качестве *единицы экстрагирования* предложение. Выбранные единицы экстрагирования ранжируются в соответствии с их весами (коэффициентами важности), обладающие наибольшими весами извлекаются и компонуются в конечный реферат. В процессе оценки единиц экстрагирования во внимание принимаются следующие факторы:

a) Важные предложения содержат слова, которые часто встречались в тексте.

b) Важные предложения содержат слова, которые использовались в заголовке или подзаголовках документа.

c) Важные предложения располагаются в начале или в конце параграфа и/или текста.

d) Важные предложения располагаются в таких позициях, которые определены соответственно жанру произведения (это специальное местоположение может быть определено автоматически с помощью обучаемого модуля).

e) Важные предложения содержат усиливающие слова (такие как «значительный», «важнейший») или определяющие фразы (например, «цель этого исследования» или «задача этой статьи»), в то время как неважные предложения содержат слова из стоп-листа (такие как «вряд ли», «неважный»).

f) Важные предложения и концепции жестко связаны между собой в рамках отдельных семантических структур.

g) Важные слова часто встречаются в тексте.

h) Важные слова являются специфическими терминами в предметной области, к которой относится документ.

i) Разные лингвистические конструкции могут обозначать один и тот же предмет действительности (учет анафорических ссылок).

Важные и неважные предложения извлекаются из исходного текста. Здесь могут возникнуть следующие проблемы: методы оценки важности единиц извлечения зачастую в значительной степени ненадежны. Простое упорядочивание извлеченных кусков текста в порядке их следования в исходном документе не гарантирует связности конечного реферата. Более того, бездумное последовательное соединение кусков текста может привести к искажению или даже потере исходного смысла предложений. Использование клауз, как единиц извлечения, добавляет еще одну проблему – проблему возможной грамматической некорректности получаемых предложений.

- ***Автоматическая генерация реферата:*** Детальный семантический анализ позволяет составить внутреннее представление исходного текста, например, в виде графа зависимостей или на основе фреймов. Затем на основе встроенных моделей и шаблонов предложений формируется конечный реферат.

Этот подход требует больших баз знаний, и сам процесс реферирования требует много времени. Такие системы значительно зависят от предметной области. Истинный глубокий семантический анализ текста на текущий момент представляется проблемным. Кроме того, из-за ограниченности генерации текста по шаблонам результирующие рефераты получаются скучными и однообразными, почти не отражают некоторых критических особенностей исходного текста. Зачастую системы, использующие автоматическую генерацию текста, накладывают неудобные (иногда - неприемлемые) ограничения на коэффициент сокращения. При условии появления полноценного семантического анализатора, приемлемого по качеству и ресурсам, возможно появление качественно новых и значительно лучших систем реферирования.

Смешанная стратегия может быть представлена методом частичной переформулировки предложений, изменения их порядка в рамках абзаца или всего реферата. Такой метод требует особо тщательного контроля над качеством реферата.

3. Постановка задачи

Целью данного проекта являлось создание программы, позволяющей автоматически составлять реферат с использованием знания о синтаксической структуре предложений исходного текста. Эта проблема возникает в случае составления кратких аннотаций, сопровождающих статьи. Обычно размер таких аннотаций строго регламентирован (четыре-пять предложений или 350-400 символов). Аннотация составляется из наиболее важных предложений текста, при

этом, чем короче и содержательнее предложение, тем лучше. Однако не всегда выбранные предложения коротки, тогда имеет смысл включить в аннотацию только наиболее важную и содержательную часть длинных предложений. Приведенная ниже система позволяет решить такую задачу.

На вход система получает текстовый документ с расставленными в нем весами слов и предложений. Программа обрабатывает текст и составляет из него реферат, сокращая длинные, но нужные предложения, и удаляя ненужные. Пользователь имеет возможность настроить коэффициент сжатия исходного текста.

4. Методы реферирования

Несмотря на все описанные трудности, несколько методов реферирования все же было успешно разработано и применено. Рассмотрим наиболее популярные из них. Очевидно, что большинство методов построено на сочетании чистых стратегий, выбранных в соответствии с представлениями разработчиков об эффективной системе реферирования. В этом разделе описывается несколько чистых стратегий:

1) TF/IDF method. *TF/IDF method (Term Frequency / Inverted Document Frequency)* – самый распространенный метод определения важности слов. Почти все системы используют его. Итак, каждому слову в предложении ставится в соответствие коэффициент его важности в тексте (вес). Он вычисляется по следующей формуле:

Частота_слова(Частота_документов/Количество_всех_документов)*

Вес предложения, соответственно, вычисляется либо как простая сумма весов его слов, либо как среднее арифметическое весов слов. *TF/IDF method* превращается просто в *TF method*, если рассматривается не коллекция документов, а отдельный документ. На вес слова так же влияют его морфологические признаки.

2) Position method. Одна из самых простых и очевидных стратегий. Основная идея – важные предложения находятся в определенных, заранее известных позициях исходного текста. В результате применения этой стратегии предложения, расположенные в обговоренных позициях текста, увеличивают свой вес на некоторый коэффициент. Обычно выбираются несколько первых предложений в каждом абзаце, несколько последних и, возможно, несколько еще дополнительных положений, детерминированных жанром.

Наиболее известная разновидность этой стратегии называется *LEAD method* и успешно применяется в системах, настроенных на реферирование газетных статей. Считается, что несколько (чаще всего 3) первых предложений – наиболее важные в каждом абзаце.

*3) **Discourse segmentation.*** Этот метод рассматривает текст как совокупность обсуждаемых в нем тем и подтем. Соответственно, строится дерево дискурса текста. Темы, которые занимают много места (включают самые длинные предложения), но несут мало информации, удаляются, пока не получится реферат нужной длины.

*4) **Sentence processing.*** Данный подход предполагает работу с предложениями исходного текста. Основная идея метода – при построении обычного реферата человек не всегда использует предложения первоисточника. Зачастую он их сокращает, убирая лишнюю информацию. Собирает два коротких предложения в одно, перефразирует и т.д. Метод предполагает аналогичное поведение системы: строится дерево предложения, далее обширные, но маловесные ветви удаляются, возможно, происходит склейка двух деревьев тем или иным способом, синтаксическая переформулировка. Реферат состоит из обработанных предложений исходного текста.

5. Общее описание системы

Зачастую в исходном тексте встречаются длинные предложения, одна часть слов которых несет большую информационную нагрузку, чем другая. Это относится, в частности, к предложениям с однородными членами. За счет всей этой лишней словарной нагрузки и большой длины предложение получает малый вес. Однако в нем могут содержаться не менее важные факты, чем в коротких предложениях. В конечном реферате хотелось бы видеть такого рода предложение, очищенным от лишних слов. Если же текст сам по себе состоит из длинных описательных предложений и коротких определений, то в реферат попадут только определения, что существенно обеднит текст. Простое перечисление фактов – это не самый лучший вариант реферата.

Нынешнее положение вещей таково, что почти все известные реализованные программы, реферирующие русскоязычные тексты, опираются на метод квазиреферирования. То есть основные попытки усовершенствования предпринимаются в направлении более точного определения весов слов и предложений исходного текста. Однако логично предположить, что наилучших результатов можно добиться, совершенствуя параллельно все аспекты программы. В противном случае, развивая лишь одну из используемых подсистем, достижению хороших результатов будет мешать явная отсталость остальных модулей программы.

Итак, в рамках этой работы была предпринята попытка сделать еще один шаг в направлении усовершенствования процесса реферирования путем дополнительной обработки предложений. В основу реализованного алгоритма положена идея реферирования

методом частичного сокращения предложений. Подающийся на вход текст обрабатывается вспомогательным модулем, в результате около каждого слова и в начале каждого предложения проставляется вес. После этого происходит частичное удаление совсем уж неважных предложений. Теперь происходит основной этап обработки оставшихся предложений. Из них выбирается наименее важное и обрабатывается реализованным в системе поверхностным синтаксическим анализатором в целях выяснения зависимости среди слов в предложении. Результат этой обработки – синтаксическое дерево зависимостей слов предложения.

После построения дерева предложения системе становится известно, каким образом можно удалить неважные слова, не нарушая согласованности предложения. То есть, если в построенном дереве предложения есть ветвь, полностью состоящая из неважных слов, то именно ее можно удалить, оставляя в предложении только важную информацию. Если возможно провести сокращение предложения, то система это делает и пересчитывает заново вес полученного нового предложения. После чего обновленное предложение возвращается на свое место в тексте. Благодаря удалению лишней информации вес предложения повышается вместе с его значимостью. Далее система снова выбирает предложение для анализа и обработки. Так происходит до тех пор, пока объем обработанного текста не достигнет величины, заданной пользователем.

Программа представляет собой однодокументное приложение для Windows. Пользователь запускает файл Reviewing.exe и открывает текстовый документ, в котором содержится исходный русскоязычный текст, или же набивает текст вручную. Процесс реферирования начинается с выбора кнопки Start в меню Actions. После предварительной обработки текста программа запрашивает желаемый коэффициент реферирования в специально вызываемом диалоговом окне. Коэффициент сжатия задается любым целым числом. После ввода требуемых данных происходит обработка текста и отображение его на экран. Пользователь имеет возможность сохранить полученный результат в отдельном или том же файле.

6. Алгоритм работы с текстом

После считывания исходного текста с экрана происходит перевод его во внутреннее представление программы с одновременным распознаванием весов предложений и слов. В рамках данной работы не предполагается разработка процесса расстановки весов в предложениях. Поэтому в качестве вспомогательного модуля используется программа, которая обеспечивает расстановку весов.

После получения от пользователя данных по коэффициенту

сжатия программа производит предварительное сокращение текста, то есть удаление полностью самых малозначимых предложений. Длиной текста считается количество входящих в него слов, которое делится на коэффициент сжатия для получения величины требуемого объема реферата. Предварительное сокращение приводит к реферату, который в полтора раза больше желаемой величины. Это позволяет избавиться от наименее важных предложений, отсутствие которых мало повлияет на информационную составляющую исходного текста. Следующий этап обработки – это основная часть данной работы, поэтому рассмотрим его подробнее.

Среди предложений предварительно обработанного текста программа выбирает то, которое имеет самый маленький вес. Следующий шаг – обработка этого предложения модулем, осуществляющим поверхностный синтаксический анализ и построение дерева зависимостей слов предложения. В соответствии с построенным деревом программа присваивает словам предложения уровни, начиная с нулевого (для сказуемого в тексте), и далее по мере продвижения вниз по дереву: уровень равен количеству ребер между обрабатываемым словом и корнем. Одним из системных параметров является минимальное значение веса слова, которое можно считать значимым (m_InText→t_controlweight). Теперь программа должна выполнить собственно сокращение. Можно пойти по пути удаления всех концевых вершин при условии, что их вес меньше пороговой величины. Или же выбрать одну ветвь, слова которой также маловесны, и удалить ее полностью. Удаление происходит до тех пор пока не встретится слово, вес которого будет больше или равен m_InText→t_controlweight, или пока уровень не станет равен m_InText→t_currlevel (значит, скорее всего, следующее слово – подлежащее или сказуемое, удаление которых неприемлемо), или пока количество оставшихся слов в предложении не станет меньше m_InText→t_ controllength (что означает, что в предложении остается минимум зависимых слов). Все эти ограничения гарантируют, что, во-первых, от предложения не останется один скелет в случае, если достаточно важными в нем окажутся только главные члены. Во-вторых, что не будет удалено ни подлежащее, ни сказуемое. В-третьих, не будут удалены важные слова. Благодаря тому, что удаление происходит в жесткой зависимости от дерева зависимостей слов предложения, решается главная проблема реализуемого метода реферирования – угроза несогласованности предложения после удаления из него части слов.

Модифицированное предложение заносится обратно в текст на место старого. Происходит пересчет текущей длины реферата в соответствии с количеством удаленных слов и пересчет веса предложения. После процедуры редукции вес предложения увеличивается из-за удаленных слов, так как алгоритм расстановки

весов предполагает значение веса предложения равным среднему арифметическому весов слов в нем. Так как удаление затрагивает лишь маловажные слова, то числитель исходной дроби практически не изменяется, а знаменатель заметно уменьшается, что определяет рост величины среднего арифметического. Таким образом, из предложения удаляются маловесные слова и остаются только нужные, и вес предложения увеличивается.

Если редукцию предложения провести не удалось, то ему присваивается маркер pm_cannotmod, который означает, что это предложение уже невозможно модифицировать, и оно должно войти в конечный реферат полностью. На следующем шаге процедура повторяется с начала. Самое «легкое» предложение ищется среди тех, которые еще можно модифицировать. Если все предложения получили маркер pm_cannotmod, то реферат считается построенным. Если после модификации очередного предложения количество слов в текущем тексте достигло значения длины требуемого реферата, то процесс завершается и текущий текст выдается в качестве реферата.

Во время обработки текста фактического удаления предложений не происходит, так как это затруднило бы отображение результата в случае выбора пользователем варианта «отображать исходный текст с пометкой слов, вошедших в реферат». В реальности предложение, помеченное на удаление из текста, получает маркер pm_ignore и при дальнейшей обработке текста игнорируется. То же самое касается слов, они получают маркер wm_delete и далее не рассматриваются.

7. Детали реализации

1) Класс O_Text.

После открытия нужного файла пользователем и выбора пункта меню Actions→Start в программе создается объект типа O_Text, который является хранилищем текущего состояния текста. Ссылка на него m_InText хранится в экземпляре текущего документа приложения. В поле t_text с помощью метода t_BuiltText() заносится список предложений, представляемых объектами класса O_Predlozh. Для маркеров конца абзаца и конца текста создается собственное предложение, состоящее из одного слова, помеченного wm_EOP или wm_EOT соответственно. Самому предложению присваивается статус pm_spec, который позволяет не рассматривать это предложение в процессе составления реферата, но помогает сохранить структуру исходного текста.

Параллельно считается количество слов в тексте (исключая пробелы, знаки препинания, маркеры конца абзаца и конца текста). Полученная величина помещается в поле t_currlength созданного

объекта. После этого программа в специальном диалоговом окне запрашивает у пользователя коэффициент сжатия. В поле t_finallength помещается значение поля t_currlength, поделенное на введенное пользователем число. После этого программа вызывает метод t_TextReduce(), который производит сокращение текста. В результате работы этого метода некоторые предложения текста получают статус pm_ignore и некоторые слова в остальных предложениях маркируются wm_delete. Обе эти пометки означают, что в реферат такие слова (предложения) не входят. Следующий шаг – применение метода t_OutPutText(), который распечатывает текст с учетом проставленных пометок.

Еще один метод класса O_Text – это метод t_FindLeastCost(). Он помогает находить в тексте предложение минимальной стоимости среди тех, что не помечены на удаление.

2) Класс O_Predlozh.

Считывание текста происходит в переменную типа CString. Входная последовательность символов разбивается по пробелам и признакам конца абзаца. Каждая выделенная часть рассматривается на предмет примыкающих к ней знаков препинания. Если таковые находятся, то они заносятся в список слов текущего предложения (p_predl) в качестве отдельных слов с пометкой wm_sign. Оставшиеся символы считаются словом и заносятся в текущее предложение под маркой wm_word. Каждое слово в предложении представляет собой объект типа O_Word. Если встреченный знак был точкой, вопросительным или восклицательным знаком, это означает, что текущее предложение закончилось. Происходит заполнение полей текущего предложения:

- p_weight – вес предложения;
- p_modifier – модификатор предложения, который используется при пересчете веса;
- p_len – количество слов в предложении (без учета знаков препинания);
- p_marker – маркер предложения (для всех принимается равным pm_canmod, кроме предложений, состоящих из признаков конца абзаца и текста, для которых он равен pm_spec);

После построения внутреннего представления текста вызывается его обработчик – t_TextReduce(), который, в свою очередь, вызывает метод p_ReduceSent(), осуществляющий работу по анализу и сокращению (если это возможно) предложения.

3) Класс O_Word.

Объектами этого класса представлены слова исходного текста. Каждое предложение содержит, помимо уже упомянутых полей, поле p_predl, представляющее собой массив указателей на объекты типа

O_Word, в которых собственно и находятся слова (символьное их представление) и их параметры:

- w_word – символьное представление слова;
- w_weight – вес слова в данном тексте;
- w_mark – текущий маркер слова;
- w_part – часть речи;
- w_params – массив остальных параметров слова, представленных последовательностью символов;
- w_master – индекс хозяина данного слова в предложении;
- w_slaves – список индексов зависимых слов в предложении;
- w_currslave – текущий индекс в массиве зависимых слов.

Также у этого класса есть метод w_SameWord(O_Word *), который помогает определить, может ли текущее слово и переданный параметр быть однородными членами предложения.

4) Класс O_SyntAnalizer.

Объект этого типа создается для каждого предложения, которое подвергается модификации. По сути своей – это поверхностный синтаксический анализатор, который имеет единственное поле – sa_pre, содержащее указатель на то предложение, что сейчас модифицируется. Основная же его роль – это контейнер методов обработки сочетаний рядом стоящих слов в предложении. Каждый метод представляет собой обработку одного шаблона отношений. Правильная последовательность применения этих методов гарантирует корректность работы анализатора.

8. Заключение

Широкое применение систем автоматического реферирования позволит существенно улучшить работу таких приложений, как поисковые системы, в которых вместо предложений со словами запроса можно в качестве результата представлять краткий реферат всего документа, что позволит объективнее оценить его и отсеять, если его основные положения не соответствуют запросу. Еще одно очевидное применение данных систем – это составление кратких аннотаций к дипломам, курсовым и статьям, позволяющим без детального прочтения составить представление о предмете работы. Подобные краткие выжимки полезно использовать в картотеках и базах данных.

Как уже говорилось, предложенный алгоритм не претендует на построение дерева предложения, лишь на построение дерева связей слов, однако в контексте требуемой задачи этого вполне достаточно. Более того, полноценный синтаксический анализатор может улучшить общую работу системы, однако потребует достаточно большого количества дополнительных ресурсов, что зачастую не оправдано.

Когда компьютеры приобретают всё большую мощность, а объем информации, необходимой для своевременной и корректной работы и интересного отдыха, растет такими быстрыми темпами, необходимо рационально использовать своё время и ресурсы, чему наилучшим образом будет способствовать возможность автоматического создания рефератов.

Литература

1. Мальковский М.Г. Диалог с системой искусственного интеллекта – М.: Русский язык, 1977.

2. Мальковский М.Г., Грацианова Т.Ю., Полякова И.Н. Прикладное программное обеспечение: системы автоматической обработки текста – М.: Издательство МГУ, 2000.

3. Полякова И.Н., Строилов Ю. Системы автоматического квазиреферирования. *Статья в тематическом сборнике №3 «Программные системы и инструменты»* – М.: Издательство МГУ, 2002.

4. Розенталь Д.Э. Русский язык – М.: Издательство МГУ, 1988.

5. Хан У., Мани И. Системы автоматического реферирования – М.: Открытые системы, №12/2000.

6. Сайт research.metric.ru, посвященный технологиям анализа и поиска текстовой информации.

7. Сайт компании «МедиаЛингва» – www.medialingua.ru

8. Сайт программы «TextAnalyst» – www.analyst.ru

9. Mani I. Summarization Evaluation: An Overview. – *Proceedings of the NTCIR Workshop 2 Meeting on Evaluation of Chinese and Japanese Text Retrieval and Text Summarization.* Tokyo, National Institute of Informatics, 2001.

10. Salton G., Singhal A., Mitra M., Buckley C. Automatic text structuring and summarization. – *Information Processing and Management*, 33(2). Mar. 1997.

11. Branimir K. Boguraev, Mary S. Neff. Discourse segmentation in aid of document summarization. – *Proceedings of the 33rd Hawaii International Conference on System Sciences*, Maui, HI, Jan. 2000.

12. Kai Ishikawa, Shinichi Ando, Akitoshi Okumura. Hybrid Text Summarization Method based on the TF Method and the LEAD Method. – *Proceedings of the Second NTCIR Workshop on Research in Chinese & Japanese Text Retrieval and Text Summarization.* National Institute of Informatics, Tokyo, May 2000.

13. Yohei Seki. Sentence extraction by tf/idf and position weighting from newspaper articles. – *NTCIR Text Summarization Challenge*, 2001.

Бурцев А.А., Владимирова Ю.С.

Средства доступа к интерфейсу Win32 API в ДССП

С созданием новой версии ДССП [1,2], ядро которой для облегчения переноса системы на другие платформы было реализовано на языке Си, удалось обеспечить функционирование ДССП в среде современных 32-разрядных операционных систем семейства MS Windows. Однако прежние средства доступа к функциям ОС, ориентированные на MS-DOS (команда INTERR), оказались совершенно не пригодны для доступа к функциям Win32 API-интерфейса [3]. Потребовалось разработать новый механизм доступа к функциям ОС с учетом нового подхода вызова таких функций из прикладной программы.

Средствам обеспечения доступа к произвольным функциям интерфейса Win32 API из ДССП-программы и посвящена данная статья. Далее описываются предлагаемые средства и демонстрируются примеры их применения.

Средства вызова произвольной функции интерфейса Win32 API

Все функции, входящие в интерфейс Win32 API, содержатся в динамических библиотеках "kernel32.dll", "gdi32.dll", "user32.dll". В программе на языке Си вызовы функций DLL-библиотек синтаксически ничем не отличаются от вызова обычных Си-функций: для вызова такой функции надо задать ее имя, а затем в скобках перечислить необходимые параметры. Например:

```
/*1) Получение номера версии */
 NomVers=GetVersion();
/*2) Совместное умножение и деление длинных целых*/
 Res= MulDiv(nNumber,nNumerator,nDenominator);
/* 3) Получение имени текущего каталога*/
 Len=GetCurrentDirectory(nBufferLength,lpBuffer);
/* 4) Вызов диалогового окна сообщения */
 Res=MessageBox(NULL,
          " Hello! This is TEST of MessageBox ",
          "DSSPMES",MB_OK);
```

Чтобы обеспечить аналогичный вызов произвольной функции интерфейса Win32 API в ДССП предлагается команда **WIN32API_**:

```
[ P1,..,Pk,k, AdrName ] WIN32API  [ Res,CCode ]
```

Перед ее вызовом сначала необходимо поместить в стек операндов параметры вызываемой функции P1,..,Pk (если они есть), причем в том же порядке, как они задавались бы при вызове соответствующей Си-функции, т.е. так, чтобы на вершине стека оказался последний параметр (Pk). Затем следует задать их количество (целое k) и имя функции (указатель на строку AdrName). Каждый параметр вызываемой функции (независимо от его типа) должен занимать в стеке одну позицию, т.е. представляться 32-битной величиной, младшая часть которой будет использована при вызове, если в качестве параметра потребуется 8-ми или 16-битная величина.

Код завершения этой команды (CCode) вместе с результатом осуществленного вызова функции (Res) останутся соответственно в вершине и подвершине стека операндов вместо всех величин, приготовленных в нем для этого вызова. Ненулевое значение кода завершения (CCode) означает, что вызов функции сделать не удалось. Причиной такой неудачи является, как правило, неправильное имя функции. (Заметим, что имя функции в DLL-библиотеке может отличаться от имени, употребляемом при демонстрации вызова этой функции в Си-программе, которая перед компиляцией будет еще подвергаться препроцессорной обработке).

В большинстве случаев, однако, непосредственно использовать рассмотренную выше команду **WIN32API_** бывает неудобно по двум причинам. Одна из них заключается в том, что после вызова этой команды каждый раз приходится явно проверять код ее завершения, чтобы удостовериться, что она исполнилась нормально. Другое неудобство этой команды связано с тем, что строки, используемые при вызовах Си-функций, должны представляться с признаком конца, тогда как обычные строки в ДССП ("...") представляются без такого признака, но снабжаются счетчиком длины.

Поэтому на основе примитива **WIN32API_** в ДССП определено слово **WIN32API**, которое само преобразовывает строку имени перед выполнением универсального вызова функции, а затем осуществляет проверку кода завершения и возбуждает исключительную ситуацию **WIN32FAIL!**, если вызов функции не состоялся. Строка имени функции перед исполнением слова **WIN32API** должна задаваться адресом и длиной (AdrName,Len), причем после самого имени в строке следует разместить еще один символ (например, пробел). После исполнения слова **WIN32API** в стеке останется только значение результата вызова заданной функции (Res):

```
[ P1,..,Pk,k, AdrName,Len ]  WIN32API  [ Res ]
```

Существенно, что слово **WIN32API** можно легко определить как обычную ДССП-процедуру:

```
: WIN32API [P1,..,Pk,k,Name,Len]\0[P1,..,Pk,k,Name]
      WIN32API_ [Res,CCode] IF- WIN32FAIL! [Res] ;
: \0 [Для преобразования строки со счетчиком длины]
[S,Len] 1- C2 + 0 <!TB [в строку с признаком конца]
[S] ; [поставим код нуля вместо последнего символа]
[ объявляется исключительная ситуация : ]
  TRAP WIN32FAIL!  WIN32FAIL!_REAC
: WIN32FAIL!_REAC [и стандартная реакция на нее]
      ." Сбой при вызове WIN32API ! " RESTART ;
```

В этом определении используется вспомогательное слово **\0**, которое обеспечивает преобразование строки со счетчиком длины в строку с признаком конца, а также может быть вызвана объявленная в ДССП исключительная ситуация **WIN32FAIL!**. Употребляя операцию **WIN32API**, следует учитывать, что если на ситуацию **WIN32FAIL!** никакой своей реакции не устанавливается, то в случае ее возникновения будет выполняться стандартная реакция системы, в результате которой напечатается сообщение "Сбой при вызове WIN32API !" и будет произведен перезапуск ДССП.

Покажем на конкретных примерах, как можно применять слово **WIN32API** для выполнения в ДССП-программах вызовов функций интерфейса Win32 API:

```
[ 1) Получение номера версии ОС ]
: GetVersion [] 0 "GetVersion " WIN32API [NomVers];
[ 2) Совместное умножение/деление длинных целых ]
: MulDiv [Num1,Num2,nDiv] 3 "MulDiv " WIN32API
                                  [Num1*Num2/nDiv] ;
```

Указатели, задаваемые при вызовах функций Win32 API, должны представлять собой реально значащие адреса в адресном пространстве самой системной программы (“dssp.exe”), создающей среду для интерпретации ДССП-программ (т.е. программ, составленных на языке самой ДССП, тела которых формируются в сшитом коде). Такие адреса будем называть реальными или системными адресами.

Адреса, используемые для обращения к ДССП-процедурам и переменным, определенным в ДССП, представляют собой смещения в адресном пространстве памяти, которое используется внутри самой ДССП. Такие адреса будем называть ДССП-адресами или внутренними адресами ДССП.

Для исполнения вызова функции Win32 API ДССП-адреса необходимо преобразовывать в реальные и наоборот. Для этого в ДССП предлагается две команды: **REALADR** и **DSSPADR**. Команда **REALADR** преобразует ДССП-адрес в реальный, а команда **DSSPADR** - наоборот, реальный адрес - в ДССП-адрес :

```
[DsspAdr]    REALADR    [RealAdr]
[RealAdr]    DSSPADR    [DsspAdr]
```

В качестве примеров вызовов функций Win32 API с применением операций преобразования адресов и строк продемонстрируем, как узнать и распечатать имя текущего каталога, и как вызвать на экран стандартное диалоговое окно (см. рис.1) с выдачей в нем короткого сообщения:

```
[ 3) Получение имени текущего каталога ]
 256 VALUE NameDirLen  NameDirLen DATABLCK NameDir
:  GetCurDir  []     NameDirLen     NameDir  REALADR
 [nBufferLength,lpBuffer] 2 "GetCurrentDirectoryA "
    WIN32API [RealLen] NameDir E2 TOS [] ;
[ 4) Вызов диалогового окна сообщения ]
 0000 VALUE MB_OK [константа стиля окна сообщения]
:            DsspMsgBox              []            NULL
   " Hello! This is TEST of MessageBox " \0 REALADR
   "DSSP_MESSAGE     "     \0     REALADR          MB_OK
   4 "MessageBoxA " WIN32API [Res] D [] ;
```

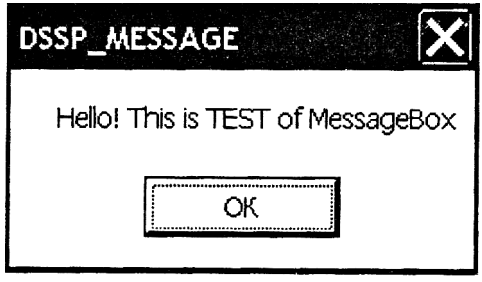


рис. 1. Пример стандартного диалогового окна сообщения

Средства обеспечения запуска оконной ДССП-процедуры

Для создания Windows-приложений с удобным графическим интерфейсом требуется составлять так называемые оконные процедуры. Оконная процедура (или процедура окна) - это особая процедура прикладной программы, которая вызывается операционной системой всякий раз, когда Windows-приложение должно отреагировать на посланное ему событие (сообщение). Поэтому подобная процедура называется процедурой обратного вызова (callback function).

По соглашениям интерфейса Win32 API [3] такая процедура (WndProc) должна принимать 4 входных 32-разрядных параметра и возвращать 32-разрядный результат. В программе на языке Си она должна быть оформлена со следующим заголовком:

```
LRESULT CALLBACK WndProc ( HWND hwnd,
          UINT iMsg, WPARAM wParam, LPARAM lParam)
```

Оконная процедура, как процедура обратного вызова, отличается от обычных процедур, и должна особым образом оформляться в ДССП-программе. (Аналогично особому оформлению процедуры обработки прерывания). Заметим, что тело оконной процедуры должно начинаться в машинном коде, так как вызываться оно будет обычной машинной командой вызова подпрограммы. Но такое тело должно запускать интерпретацию ДССП-процедуры, назначенной в ДССП-программе в качестве процедуры окна.

Для формирования подобного тела требуется зарезервировать память объемом 14 байт и исполнить команду **FORMWNDPROC**:

```
[WPA] FORMWNDPROC WNDPROC [RealWPA]
```

Для этой команды следует в вершине стека задать адрес памяти (WPA), где будет формироваться тело запуска, а следующим словом назначить имя оконной ДССП-процедуры (WNDPROC), интерпретацию которой это тело запустит. После исполнения этой команды в вершине стека останется реальный адрес (RealWPA) той области памяти, где будет сформировано тело запуска оконной процедуры. В дальнейшем при запуске созданного приложения полученный реальный адрес понадобится сообщить операционной системе в качестве адреса процедуры окна данного приложения.

В ДССП предлагается и другая команда **_FORMWNDPROC** формирования тела запуска, для которой адрес процедуры окна (AdrWNDPROC) задается с стеке:

```
: FORMWNDPROC [WPA] GTP [WPA,AdrWNDPROC]
                    FORMWNDPROC [RealWPA] ;
```

Предыдущая команда предлагает более удобный способ задания процедуры окна и определяется через данную следующим образом:

```
: FORMWNDPROC [WPA] GTP [WPA,AdrWNDPROC]
                    FORMWNDPROC [RealWPA] ;
```

Для удобного резервирования блока памяти под формируемое тело запуска предлагается команда **WNDPROCAREA**, которая отводит в памяти 14-байтный блок с указанным следом именем (WPANAME), вызов которого будет засылать в стек ДССП-адрес выделенного блока:

```
WNDPROCAREA  WPANAME
```

ДССП-процедура, задаваемая в команде **FORMWNDPROC** в качестве процедуры окна, должна потреблять из стека операндов 4 входных параметра, которые задаются оконной процедуре при вызове (в том же порядке, как они перечислены в заголовке Си-функции), и оставлять вместо них один, который будет возвращен как результат вызова оконной процедуры. Поэтому определение ДССП-процедуры окна должно иметь следующий вид :

```
: WNDPROC [hwnd,iMsg,wParam,lParam] ... [lResult] ;
```

Пример построения оконного приложения в ДССП

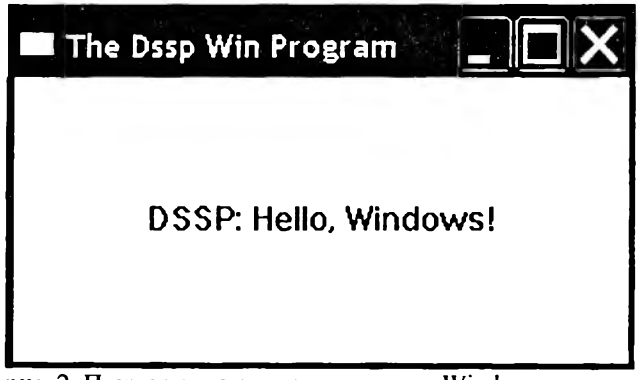


рис. 2. Пример окна самостоятельного Windows-приложения

Для демонстрации возможностей ДССП по созданию Windows-приложений на базе интерфейса Win32 API далее предлагается пример построения ДССП-программы, которая в среде 32-разрядной ОС Windows (95, 98, 2000, XP) обеспечивает функционирование самостоятельного приложения со своим отдельным окном (см. рис. 2), аналогичного тому, что представлено в виде простейшей программы приветствия (Hello,world) на языке Си в [4, гл.1].

```
      [ПРИМЕР ПРИМЕНЕНИЯ ИНТЕРФЕЙСА WIN32API ]
LOAD WINAPI.dsp [ Загрузка определений WIN32 API ]
PROGRAM $DSSPWIN
CR ." DSSPWIN : DSSP-приложение со своим окном "
      [--- ПЕРЕМЕННЫЕ основной процедуры ---]
 HINSTANCE  VAR hInstance [ Дескриптор приложения ]
 HWND VAR hwnd   [ Дескриптор Окна ]
 MSG  VAR msg    [ Сообщение ]   : &msg msg REALADR ;
 WNDCLASSEX VAR wndclass [ Класс Окна ]
             : &wndclass wndclass REALADR ;
LONG VAR szAppName [указ-ль на строку имени класса]
 : !szAppName! [ Инициализация имени класса окна ]
    "dsspwin " \0 REALADR ! szAppName ;
 : !hInstance [Инициал-ция дескриптора приложения]
  NULL 1 "GetModuleHandleA " WIN32API ! hInstance ;
 : Initwndclass [Инициал-ция структуры класса окна]
     !0 wndclass [ обнуление всей структуры ]
         [ Установка отдельных ее полей : ]
     WNDCLASSEXlen  wndclass ! .cbSize
     CS_HREDRAW CS_VREDRAW | wndclass ! .style
     FormDsspWndProc wndclass ! .lpfnWndProc
     hInstance wndclass ! .hInstance
     NULL IDI_APPLICATION 2 "LoadIconA "
               WIN32API wndclass ! .hIcon
     NULL IDC_ARROW 2 "LoadCursorA "
               WIN32API wndclass ! .hCursor
     WHITE_BRUSH 1 "GetStockObject "
               WIN32API wndclass ! .hbrBackground
     NULL IDI_APPLICATION 2 "LoadIconA "
               WIN32API wndclass ! .hIconSm
     NULL wndclass ! .lpszMenuName
     szAppName wndclass ! .lpszClassName  ;
 : RegisterClass [ Регистрация класса окна ]
     &wndclass 1 "RegisterClassExA " WIN32API D ;
 : CreateWindow [ Создание окна ] 0
```

```
     szAppName [ Имя класса Окна ]
     "The Dssp Win Program " \0 REALADR [Заголовок]
     WS_OVERLAPPEDWINDOW [ стиль окна ]
     CW_USEDEFAULT C C C [xpos,ypos,xsize,ysize]
     NULL          [ родительское окно(Нет) ]
     NULL          [ window menu handle (Пока нет)]
     hInstance     [ program instance handle ]
     NULL [ук-ль на доп.параметры окна (пока нет)]
  12 "CreateWindowExA " WIN32API [hwnd] ! hwnd [ ];
: ShowWindow  [ Показать окно ]
   hwnd SW_SHOWDEFAULT 2 "ShowWindow " WIN32API D ;
: UpdateWindow hwnd 1 "UpdateWindow " WIN32API D ;
: GetMessage  [Получить сообщение, 0 - Конец цикла]
    &msg NULL 0 0 4 "GetMessageA " WIN32API ;
: TranslateMessage
    &msg 1 "TranslateMessage " WIN32API D ;
: DispatchMessage
    &msg  1 "DispatchMessageA "  WIN32API D ;
:: : DSSPWIN [ ] [----- ОСНОВНАЯ ПРОЦЕДУРА ------]
    !szAppName!  [ Установка имени класса окна ]
    !hInstance [Инициал-ция дескриптора приложения]
    Initwndclass [Иниц-ция структуры класса окна]
    RegisterClass [ Регистрация класса окна ]
    CreateWindow  [ Создание окна ]
    ShowWindow    [ Высвечивание окна ]
    UpdateWindow  [ Обновление окна ]
        [ цикл обработки сообщений : ]
    GetMessage DW HandleMsg
    msg .wParam [ итоговый результат] ;
   : HandleMsg TranslateMessage DispatchMessage ;
WNDPROCAREA DsspWPArea [ Резервирование области
          памяти для тела запуска процедуры окна ]
 : FormDsspWndProc [ Формирование тела запуска ]
    DsspWPArea FORMWNDPROC DsspWndProc [RealAdr] ;
 [--- ПЕРЕМЕННЫЕ ОКОННОЙ процедуры ---]
  HDC VAR     hdc
  PAINTSTRUCT VAR ps    : &ps ps REALADR ;
  RECT        VAR rect  : &rect rect REALADR ;
[---------- ПРОЦЕДУРА ОКНА -------------------]
:: : DsspWndProc [ hwnd, iMsg, wParam, lParam ]
     C3 BR WM_CREATE  Handle_Msg
           WM_DESTROY Handle_WM_Destroy
           WM_PAINT   Handle_WM_Paint
           ELSE  DefWindowProc      [LResult] ;
```

```
  : Handle_Msg DDDD 0 ;
  : DefWindowProc  [ hwnd, iMsg, wParam, lParam ]
        4 "DefWindowProcA "  WIN32API [LResult] ;
  : Handle_WM_Destroy DDDD 0 1 "PostQuitMessage "
        WIN32API D 0 [0] ;
  : Handle_WM_Paint [ hwnd, iMsg, wParam, lParam ]
        BeginPaint ! hdc GetClientRect
        "DSSP: Hello, Windows! " DrawText
        EndPaint Handle_Msg [0] ;
  : BeginPaint C4 [hwnd] &ps 2 "BeginPaint "
        WIN32API [hdc] ;
  : GetClientRect C4[hwnd] &rect 2 "GetClientRect "
        WIN32API D [] ;
  : DrawText [Txt,len] \0 REALADR hdc E2 -1 &rect
          DT_SINGLELINE DT_CENTER | DT_VCENTER |
          5 "DrawTextA " WIN32API D [] ;
: EndPaint C4[hwnd] &ps 2 "EndPaint " WIN32API D[];
```

Заключение

Предложенные средства, обеспечивающие доступ к функциям интерфейса Win32 API из ДССП-программ, открывают возможность построения в ДССП качественно новых диалоговых программ, способных осуществлять взаимодействие с пользователем не только в консольном режиме, но и в режиме окна.

Литература

1.Брусенцов Н.П., Захаров В.Б., Руднев И.А., Сидоров С.А. Диалоговая система структурированного программирования ДССП-80.- В кн.: Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ,1986, с.3-21.

2.Бурцев А.А., Франтов Д.В., Шумаков М.Н. Разработка интерпретатора сшитого кода на языке Си. // Вопросы кибернетики. Сб. статей под ред. В.Б. Бетелина. - М., 1999, с.64-76.

3.Рихтер Дж. Windows для профессионалов: Программирование для Windows 95 и Windows NT 4 на базе Win32 API. – М.: Изд. отдел "Русская редакция " ТОО "Channel Trading Ltd.", 1997.

4.Петзолд Ч. Программирование для Windows 95. – Спб.: BHV-Санкт-Петербург, 1997.

Мещеряков Д.К.

Простой способ генерации случайных чисел с использованием примитивов ОС[6]

Введение

Существует большое число приложений, использующих в своей работе случайные числа и их последовательности. Традиционно вместо случайных чисел используются так называемые псевдослучайные числа, получаемые с помощью специальных функций. Рассматривается способ получения чисел, действительно обладающих случайной природой, основанный на использовании счетчика тактов центрального процессора ЭВМ и функций задержки, предлагаемых операционной системой.

Области применения

Случайные числа применяются в разнообразных приложениях, в том числе:

- При имитационном моделировании – для генерации потоков и значений свойств событий. В качестве примера можно привести моделирование процессов стрельб или генетические алгоритмы ([7]). Здесь обычно достаточно псевдослучайных чисел при условии, что распределение их последовательности отвечает нужным требованиям, например, является нормальным с заданными параметрами, однако в некоторых случаях могут требоваться случайного характера отклонения свойств получаемых чисел в целях вариации хода моделирования, например, для имитации внешних воздействий на популяцию в генетическом алгоритме.
- В игровых приложениях – для генерации начальных раскладок колод карт, определения исходов стрельб боевых единиц или действий компьютерного противника. Если речь идет о домашних развлечениях, то особо жестких требований к случайным числам не предъявляется, но в коммерческих приложениях, например, в игровых автоматах, требования могут быть очень жесткими.

[6] Работа выполнена при поддержке гранта РФФИ № 02-07-90130.

- В криптографии, вопросах аутентификации и других смежных вопросах – для генерации ключей, паролей и других аналогичных параметров (подробнее, например, в [6]). Здесь требования к случайности чисел могут быть исключительно жесткими.

В тех случаях, когда используются псевдослучайные числа, все равно необходимы начальные значения, которые в одних приложениях можно задавать заранее, а в других очень желательно или необходимо получать случайными, а зачастую и обладающими определенными свойствами, в т.ч. статистическими характеристиками.

Описываемый подход может использоваться для получения случайных чисел в приложениях, где потребность в случайных числах невелика – для инициализации генераторов псевдослучайных чисел или при генерации паролей при условии, что его реализация действительно обеспечивает генерацию последовательностей, удовлетворяющих требованию конкретных приложений.

Существующие решения

К настоящему моменту действительно случайные числа получают тремя основными способами:

- С помощью аппаратных (так называемых шумовых) датчиков. Это аппаратные устройства, периодически измеряющие параметры функционирования специальных электрических схем, входящих в их состав. Какие именно - выбирается из расчета, чтобы их значения имели случайную природу. Из реализаций можно упомянуть Intel RNG – компонент одного из наборов микросхем для системных плат ПЭВМ. Обычно аппаратные датчики выдают последовательности очень хорошего качества, но у них есть очевидные недостатки: они дороги и сложны в установке.
- Считыванием значения системного таймера. Таймер обнуляется при запуске ЭВМ и периодически увеличивается на единицу (см. [5]). Считывание выполняется обычно с помощью функций операционной системы. Этим способом можно получить, например, одно начальное значение для генератора ПСЧ в начале моделирования. Способ имеет ряд недостатков: разрядность регистра системного таймера невелика, невозможно получить подряд несколько действительно случайных чисел, в ряде случаев можно достаточно достоверно предсказать значение, которое будет получено с таймера, с помощью программных средств.
- Анализом вводимой пользователем информации. Пользователю предлагается «понажимать клавиши» или подвигать мышью. Траектории движения или интервалы между нажатиями считаются случайными. Очевидный недостаток: необходим пользователь,

процедура ввода данных занимает длительное время и может раздражать пользователя.

Описываемое решение не требует пользователя, позволяет получать порядка сотен байт случайных чисел в секунду и работает на ЭВМ с ЦП Intel Pentium, AMD K5 или более поздних моделей без дополнительного оборудования.

Суть метода

Процессоры Intel Pentium, AMD K5 или более поздние модели оснащены счетчиком тактов – специальным 64-разрядным регистром, обнуляемым при запуске процессора и увеличиваемым автоматически с каждым тактом процессора. В процессорах других архитектур обычно имеются аналогичные средства. Счетчик тактов программно доступен с помощью специальной инструкции процессора, копирующей его значение в регистры общего назначения. Накладные расходы на такой доступ составляют всего несколько процессорных тактов, а выполнять инструкцию могут любые пользовательские или системные программы. Младшие разряды счетчика циклически изменяются с очень высокой скоростью (важно помнить, что старшие разряды меняются очень медленно). Очевидно, прочитав значение счетчика тактов в некоторый момент времени и взяв несколько (сколько именно – не всегда тривиальный вопрос) младших его разрядов, мы получим случайное число.

Исходный текст для такого чтения может выглядеть, например, так:

```
{
   char rnd;
   asm
   {
      rdtsc;//прочитать значение в регистры edx:eax
      mov byte ptr rnd, al;//rnd получает случайное значение
   };
};
```

Идея не является новой и часто упоминается в различных сетевых конференциях, посвященных вопросам программирования. В то же время уделяется слишком мало внимания деталям реализации и качеству получаемых чисел. Зачастую предлагается даже брать все 64 разряда, что в большинстве случаев недопустимо, ибо значения старших разрядов легко предсказуемы – самые старшие остаются нулевыми в течение многих недель работы ПЭВМ.

Если теперь использовать какой-либо источник событий, слабо связанных или вовсе не связанных с тактированием процессора, и

связать чтение счетчика тактов с такими событиями, то можно получить последовательность случайных значений. В качестве такого источника можно использовать планировщик операционной системы и функции задержки. Функции задержки дают указание планировщику приостановить выполнение потока команд на время не менее заданного. Планировщик использует в своей работе системный таймер, который обычно тактируется отдельной от центрального процессора частотой, накладные расходы на переключение потоков труднопредсказуемы, число потоков и их потребности в процессорном времени часто и труднопредсказуемо изменяются, поэтому конкретные значения задержек в терминах числа процессорных тактов предсказать исключительно сложно. Использовать аналогичным способом системный таймер было бы крайне нежелательно, поскольку он же используется и для реализации задержек, а для счетчика тактов связь между длиной задержек и числом прошедших тактов намного сложнее.

Достоинства данного подхода очевидны: низкая стоимость реализации на современных ПЭВМ, возможность получать относительно большое число случайных чисел подряд, отсутствие необходимости в пользователе.

Реализация и результаты

Для исследования получаемых чисел была разработана программа, работающая в среде Win32 и генерирующая последовательность с помощью следующей процедуры (псевдокод имитирует язык Си):

```
while(1)
{
  Sleep(1);//задержка
  newrnd();//получить очередное число из счетчика тактов
};
```

Функция Sleep() из набора системных функций Win32 приостанавливает программу не менее чем на заданное число миллисекунд (в данном случае – на одну).

Таким способом были сгенерированы подряд 32768 байт (время генерации – 125 секунд вместо ожидаемых 33, что показывает низкую точность реализации задержек операционной системой), и над этим набором проведены простейшие тесты. Детальное рассмотрение процессов тестирования проводится, например, в [1].

Тест на равномерность проверяет, является ли распределение чисел равномерным. Отрезок [0..255] разбивается на восемь равных отрезков, и определяется число попаданий в каждый из таких отрезков. Для равномерно распределеной на отрезке [0..255] случайной величины

вероятность принять значение, принадлежащее любому из отрезков, составляет 1/8. Для проверки гипотезы о равномерности распределения чисел в исследуемом наборе применяется критерий согласия Пирсона - вычисляется величина χ^2 и по таблице для распределения χ^2 с семью степенями свободы (например, из [2]) определяется доверительная вероятность.

Из таблицы для ДВ = 0.8	3.82
Из таблицы для ДВ = 0.7	4.67
Вычислено	3.87

Таблица 1. Тест на равномерность.

В соответствии с критерием согласия Пирсона, числа можно считать равномерно распределенными с доверительной вероятностью не менее 0.7.

Дополнительно был проведен тест на равномерность распределения значений младшего разряда, табличные значения взяты для χ^2 распределения с одной степенью свободы.

Из таблицы для ДВ = 0.7	0.15
Из таблицы для ДВ = 0.5	0.45
Вычислено	0.26

Таблица 2. Тест на равномерность для младшего разряда.

В соответствии с критерием согласия Пирсона, значения младшего разряда можно считать равномерно распределенными с доверительной вероятностью не менее 0.5

Тест на случайность проверяет, является ли появление единиц и нулей в разрядах чисел равновероятным. Двоичные записи чисел считаются строками матрицы, и определяется число единиц в каждом столбце. Теоретически вероятность появления единицы в любом разряде любого числа составляет ½. Аналогично предыдущему случаю применим критерий согласия Пирсона, табличные значения берутся для χ^2 распределения с восемью степенями свободы.

Из таблицы для ДВ = 0.9	3.49
Из таблицы для ДВ = 0.8	4.59
Вычислено	4.33

Таблица 3. Тест на «случайность».

В соответствии с критерием, числа можно считать случайными в вышеуказанном смысле с доверительной вероятностью не менее 0.8.

Предложенная реализация является исключительно простой. В качестве примера ее использования можно привести игру в «русскую рулетку» с восьмизарядным револьвером. Ее исходный текст состоит лишь из нескольких строк – необходимо только сгенерировать случайный байт, обнулить старшие пять разрядов и проверить, равен ли результат нулю (или любому другому целому от нуля до семи по вкусу

программиста). После этого можно выводить результат попытки на консоль или любым другим способом.

Заключение

Описан способ генерации случайных чисел и их последовательностей, допускающий полностью программную реализацию, не требующий участия пользователя и позволяющий генерировать порядка сотен байт случайных чисел в секунду. Качество получаемых чисел может быть приемлемым для многих приложений, но этот вопрос необходимо тщательно исследовать применительно к каждому конкретному приложению.

Автор благодарит своих друзей и коллег за участие в тестировании программного обеспечения, основанного на использовании рассмотренного подхода.

Литература

1. Кнут Д. Искусство программирования. Т. 2. Получисленные алгоритмы. – М.: Издательский дом «Вильямс», 2000.
2. Ивашев-Мусатов О.С. Теория вероятностей и математическая статистика. - М.: ФИМА, 2003.
3. Юров В. Assembler: практикум – СПб: Питер, 2001.
4. Юров В. Assembler: специальный справочник. – СПб.: Питер, 2001.
5. Гук М. Аппаратные средства IBM PC. Энциклопедия, 2-е изд. – СПб.: Питер, 2001.
6. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2001.
7. Стариков А. Генетические алгоритмы – математический аппарат http://www.basegroup.ru/genetic/math.htm

Герасимов С.В., Машечкин И.В., Чжао А.В.

Концепции и архитектура медицинской информационно-аналитической системы

1. Введение

1.1. Медицинские информационные системы: структура, проблемы, актуальные задачи

В настоящее время в области медицинской информатики остро стоит вопрос создания комплексных информационных систем, предоставляющих функции как автоматизации бизнес-процессов[7] лечебно-диагностических учреждений, так и накопления, и последующего анализа собранных данных с целью выявления скрытых закономерностей, построения прогностических моделей и обеспечения поддержки в принятии решений практикующим врачам.

Как правило, медицинская информационная система решает три группы задач, распределенных по соответствующим подсистемам:

1. управление финансово-хозяйственной деятельностью;
2. автоматизация ведения историй болезни либо амбулаторных карт;
3. анализ данных.

Задачи 1-й группы затрагивают аспекты автоматизации бухгалтерской отчетности, вопросы учета материальной базы медицинского учреждения: расходных материалов и оборудования.

Подсистема автоматизации ведения историй болезни и амбулаторных карт призвана обеспечить ввод, хранение и доступ к информации, связанной непосредственно с лечебно-диагностическим процессом. Данная подсистема, как правило, является неотъемлемой частью медицинской информационной системы и предназначена для автоматизации всех аспектов ведения историй болезни и амбулаторных карт. В литературе она также известна как подсистема (либо отдельная система) ведения *электронной истории болезни (ЭИБ)*. В англоязычных источниках используется термин *Electronic Patient Record (EPR)* либо *Electronic Health Record (EHR)* system. Основными требованиями к системе ЭИБ являются:

- обеспечение автоматизации бизнес-процессов, связанных с лечением пациента;
- устойчивость к изменениям структуры потоков данных.

---

[7] Под бизнес-процессом подразумевается производственная деятельность в рамках организации.

Разработка систем ЭИБ, удовлетворяющих предъявленным требованиям, является актуальной задачей медицинской информатики. Современный уровень развития таких научных направлений как теория вероятности и математическая статистика, искусственный интеллект, повсеместное внедрение IT-технологий сделали возможным извлечение знаний, содержащихся в БД информационных систем, и последующее их использование в научно-исследовательской и практической деятельности. В медицинских информационных системах указанные возможности реализуются в подсистемах анализа данных. Актуальной проблемой, возникающей при создании подобных подсистем, является разработка и реализация специализированных алгоритмов анализа данных, предназначенных для эффективного решения медицинских научно-практических задач. В качестве примеров подобных задач можно привести как задачи, связанные с поддержкой принятия решений в практической медицине - первичная диагностика заболеваний, прогнозирование успешности оперативного вмешательства, так и задачи, возникающие в научно-исследовательской деятельности и имеющие нечеткую формулировку обнаружения знаний в накопленных данных историй болезни: кластеризация пациентов по некоторой группе признаков, обнаружение скрытых закономерностей.

1.2. Развитие медицинских информационных систем

Медицинские информационные системы получили повсеместное применение в западных лечебно-диагностических учреждениях: госпиталях и клиниках. Это связано в первую очередь с распространением страховой медицины, требующей комплексного подхода к сбору всей информации, сопутствующей лечебному процессу. Имеется значительное число зарубежных разработок в области создания экспертных систем медицинской диагностики[8]. Следует отметить, что успех западных медицинских информационных систем во многом связан со стандартизацией функциональности и протоколов обмена информации в данных системах. В качестве примеров подобных стандартов [1] можно привести DICOM[2] и HL7[3].

Большинство существующих российских программных разработок медицинских информационных систем обладают двумя существенными недостатками, первый из которых, - отсутствие комплексного подхода к построению системы, что приводит к сужению области применения и невозможности расширения системы без допрограммирования в случае изменения структуры потоков данных медицинского учреждения. В

[8] Обзор медицинских экспертных систем не входит в цели настоящей статьи, ссылки на популярные зарубежные экспертные системы можно найти здесь: http://www.computer.privateweb.at/judith/special_field3.htm

качестве примеров успешных отечественных проектов в этой сфере следует привести систему «Эверест» [4] и технологию MEDSET [5], в которых просматривается комплексный подход к автоматизации бизнес-процессов.

Вторым недостатком является отсутствие либо минимальные возможности встроенных функций анализа данных, чаще всего сводящихся к вычислению статистических характеристик выборок, представленных в базе данных. Указанные функции полезны исключительно представителям администрации больниц и поликлиник для оценки эффективности работы учреждения. В тех случаях, когда требуется более глубокий анализ данных, что происходит, как правило, в научно-исследовательских задачах, обычно используются статистические пакеты, например, STATISTICA[6].

1.3. Цели работы

Данная работа посвящена проблемам создания комплексных медицинских информационно-аналитических систем, удовлетворяющих требованиям адаптации к изменениям структуры потоков данных и наличия средств глубокого анализа данных, и ориентированных на применение в научно-практических центрах медицины для исследовательских целей, связанных с обнаружением полезных знаний в исторических данных обследований пациентов и последующего использования найденных закономерностей при разработке новых методов лечения.

В рамках проводимой работы была разработана архитектура и осуществляется программная реализация экспериментальной медицинской информационно-аналитической системы SKLIFF. Разработка ведется Лабораторией технологий программирования ВМиК МГУ им. М.В.Ломоносова совместно с Московским городским центром трансплантации печении НИИ СП им. Н.В.Склифосовского [7] в содружестве с компанией Advanced Algorithms [8].

Настоящая статья описывает системный аспект работы: архитектуру и основные концепции SKLIFF.

2. Основная часть

Деятельностью центра трансплантации является научно-практическая работа, связанная с операциями на печень. При этом большое внимание специалистами центра уделяется именно научной составляющей, нацеленной на разработку и последующее применение новых методик проведения операций (прежде всего резекций и трансплантаций печени) и ухода за пациентом в послеоперационный период. Для повышения эффективности оперативных вмешательств наряду с инновационными

чисто медицинскими решениями специалистами центра предпринимались попытки применять базовые статистические методы для выявления различного рода закономерностей и особенностей лечебного процесса, влияющих на ближайший исход операции и послеоперационное состояние пациента. Однако эти попытки затруднялись по ряду причин, основные из которых: отсутствие данных о пациентах в электронном виде и неэффективность статистических методов анализа выборок небольшого объема (100-200 историй болезни центра).

2.1. Требования к системе. Цели разработки

На начальном этапе разработки совместно с представителями центра трансплантации были выявлены требования, предъявляемые к разрабатываемой системе:

- наличие подсистемы ведения ЭИБ;
- поддержка сложноструктурированных данных;
- возможность изменения структуры потоков данных системы ведения ЭИБ без перекомпилирования исходных текстов;
- поддержка вычисляемых полей и зависимостей между анализами и данными исследований;
- обеспечение конфиденциальности данных;
- разграничение прав доступа к данным;
- наличие интегрированной подсистемы анализа данных, включающей в себя алгоритмы выявления скрытых закономерностей в выборках небольшого объема, а также допускающей расширение путем подключения новых алгоритмов анализа.

Таким образом основной целью, преследуемой при разработке, было создание комплексной информационно-аналитической системы, включающей в себя подсистемы ЭИБ и анализа данных, обладающей широкой областью применения (не ограниченной центром трансплантации печени), решающей задачу выявления скрытых закономерностей в выборках небольшого объема и допускающей возможность интеграции в систему новых аналитических методов.

2.2. Структура истории болезни

Под *системой ЭИБ* подразумевается комплекс программно-аппаратных средств, позволяющих в идеале полностью отказаться от использования неэлектронных носителей информации в лечебно-диагностическом процессе [5] кроме тех случаев, когда наличия бумажного документа требует действующее законодательство РФ.

Таким образом подсистема ЭИБ должна предоставить возможность автоматизации ведения историй болезни пациентов, находящихся на лечении в центре трансплантации печени либо любом другом лечебном учреждении.

Рисунок 0. Структура истории болезни

История болезни (рис. 1) представляет собой последовательность *событий*, обладающих временными метками. Первое событие в истории – поступление пациента, последнее – выписка. Все остальные события делятся на две группы: получение информации о текущем состоянии пациента и воздействие на пациента. К первой группе событий относятся данные анализов и осмотров пациента. Вторая группа – оперативное вмешательство либо назначение лекарственных средств.

2.3. Шаблоны событий

При проектировании системы было введено понятие *шаблона события (или просто шаблона)* – фиксированного набора атрибутов. Каждый *атрибут* шаблона события определяется именем и *доменом*, определяющим тип данных (текстовый, целый, вещественный, перечисление) и ограничение на диапазон значений атрибута.

При более глубоком изучении структуры данных центра трансплантации было обнаружено, что некоторые разновидности событий имеют иерархическую организацию. В частности, вирусологический мониторинг, кроме «линейных» атрибутов, таких как дата проведения, наличие инфекционных осложнений, содержит поля, определяющие субстраты производимых посевов, набор которых определяется индивидуально для каждого пациента. Причем по каждому субстрату в событии должны хранится выделенные разновидности микрофлоры с указанием характеристик по каждой разновидности.

Для того, чтобы обеспечить возможность работы с событиями такой структуры термин шаблона события был расширен до произвольного дерева атрибутов (рис. 2). Для поддержки атрибутов, содержащих

несколько значений (например, атрибут «субстрат посева» в вирусологическом мониторинге), используется соответствующий признак «множественности» атрибута.

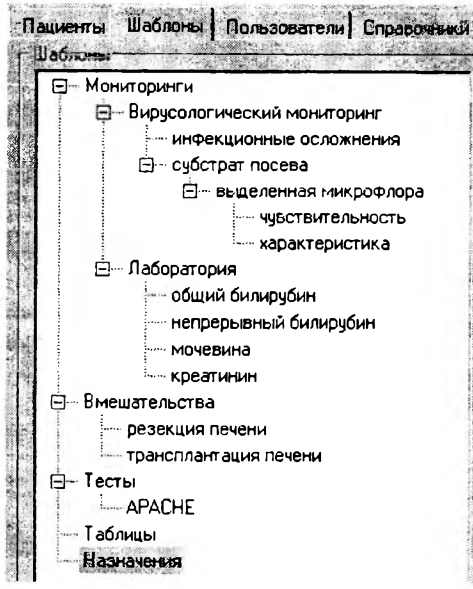


Рисунок 0. Шаблоны

В послеоперационный период пациентам центра назначается один из стандартных наборов восстановительных препаратов в зависимости от типа операции и наличия и разновидности послеоперационных осложнений. При этом сам набор, дозы и периодичность приема могут варьироваться. Для автоматизации процесса назначения препаратов существует возможность привязки к любому шаблону событий нескольких именованных вариантов заполнения значений атрибутов по умолчанию. Благодаря этому система предоставляет возможность создавать различные шаблоны назначений, каждый из которых может быть изменен при заполнении.

2.4. Вычисляемые поля

Одними из важнейших критериев, позволяющих оценить состояние пациента, страдающего заболеванием печени, являются значения баллов Apache II, Child-Turcotte-Pugh, PELD, MELD. Аналогичные системы баллов существуют и в других областях медицины. Баллы рассчитываются на основании значений показателей анализов и результатов осмотра пациента и позволяют судить о степени тяжести текущего состояния.

Для поддержки возможности учета баллов в системе и автоматизации их расчета в системе используется язык описания формул, аналогичный метаязыку, применяемому в Microsoft Excel для обеспечения связей между ячейками таблиц. Это позволяет создавать в шаблонах событий атрибуты, значения которых вычисляются автоматически и могут содержать ссылки на значения атрибутов как данного, так и других шаблонов. В случае ссылки на значение атрибута другого шаблона

события используется значение из самого последнего зависимого события.

2.5. Разграничение прав доступа

При проектировании системы учтено требование гибкого разграничения прав доступа к событиям и функциям системы. Все пользователи системы соотнесены с группами пользователей. Каждой группе может быть определен набор прав на функции системы и доступ к определенным событиям истории болезни. Среди прочих существуют права на просмотр / изменение значений событий и редактирование пользователей и прав доступа.

Вся функциональность ЭИБ разделена на 2 раздела: пользовательский и администраторский. Пользовательский раздел включает в себя функции по ведению историй болезни: просмотр, добавление, редактирование событий в истории болезни, отображение и печать на принтер отчетов динамики изменения показателей. Администраторская часть позволяет редактировать шаблоны событий и управлять правами пользователей системы.

2.6. CHDL

Для описания структуры ЭИБ разработан специализированный язык описания структуры истории болезни *CHDL (Case History Definition Language)*, основанный на XML. Организация данных на основе технологии XML также использовалась в других системах ведения ЭИБ, например, в [9]. CHDL позволяет задавать отношения между всеми сущностями системы ведения ЭИБ, как-то: пациент, донор, реципиент, событие. В конфигурационном файле, написанном на CHDL (рис. 3), администратор системы может определить внешний вид оконных форм, отображаемых при редактировании событий истории болезни.

Благодаря этим возможностям система становится более удобной в использовании и значительно расширяется круг ее применения, поскольку она может быть настроена на структуру бизнес-процессов практически любого медицинского лечебно-диагностического учреждения. Следует отметить, что CHDL является относительно простым языком, и его освоение не занимает много времени и не требует от администратора системы знаний из области программирования.

```
<!-- История болезни -->
- <object class="CaseHistory">
    <!-- Foreign Keys -->
    <field name="client_id" friendlyName="" type="int" />
    <!-- Common fields -->
    <field name="date_of_hospitalization" friendlyName="Дата госпитализации" type="datetime" nulls="true" />
    <field name="date_of_result" friendlyName="Дата выписки (смерти)" type="datetime" nulls="true" />
    <field name="result" friendlyName="Исход" type="enum" enumName="EResult" nulls="true" />
    <!-- Ссылки на другие объекты БД -->
    <link kind="many2one" class="Client" myField="client_id" hisField="id" />
    <link kind="one2many" class="Operation" myField="id" hisField="history_id" />
    <link kind="one2many" class="MainDiagnosis" myField="id" hisField="history_id" />
    <link kind="one2many" class="Analysis" myField="id" hisField="history_id" />
  </object>
```

Рисунок 0. Фрагмент конфигурационного файла CHDL

2.7. Сбор исторических данных

После начала эксплуатации пилотной версии системы возникла проблема сбора исторических данных, уже накопленных в бумажных историях болезни, поскольку этот процесс является чрезвычайно рутинным и длительным. Для его оптимизации был создан отдельный модуль сбора в виде приложения на базе Microsoft Access. В модуле используется минимальный набор значимых полей истории болезни, который был выработан совместно с представителями центра трансплантации (рис. 4). Основным требованием, предъявляемым к модулю, стали эргономичность пользовательского интерфейса и возможность максимально оперативно вводить историческую информацию. После сбора информация может быть перенесена из данного модуля в основную СУБД с помощью выполнения экспортирующего скрипта. Кроме этого данный модуль предоставляет базовые возможности системы ЭИБ и может быть использован в медицинских учреждениях с минимальной компьютерной инфраструктурой, поскольку не является требовательным к аппаратуре.

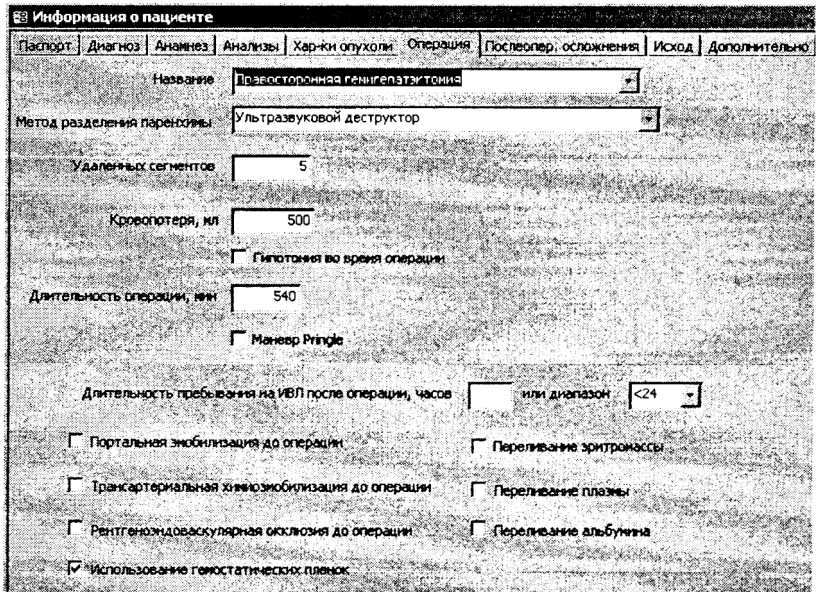


Рисунок 0. Модуль сбора исторической информации. Форма протокола операции резекции

2.8. Аналитическая подсистема – выявление скрытых зависимостей

Для реализации аналитических функций в системе используются методы и технологии *интеллектуального анализа данных (ИАД)*. ИАД - бурно развивающееся в последнее десятилетие научное направление, связанное с выявлением скрытых, заранее неизвестных и потенциально полезных знаний в базах данных [10]. ИАД предоставляет широкий набор методов, включающих методы искусственного интеллекта, теории вероятности и математической статистики, нейронных сетей и генетических алгоритмов, для решения задач анализа данных.

Разрабатываемая система предоставляет врачу-исследователю инструментарий для обнаружения скрытых зависимостей в виде *числовых ассоциативных правил (ЧАП)* [11] в данных, накопленных в историях болезни. Попытки обнаруживать ассоциативные правила в медицинских данных уже предпринимались в работах [12] и [13]. Обнаруженные зависимости смогут помочь оптимизировать методику проведения оперативных вмешательств и лечения в послеоперационный период. В настоящий момент ведутся научно-исследовательские работы по разработке специализированных алгоритмов выявления ЧАП в

условиях «широкой» (с большим числом атрибутов) выборки относительно небольшого объема (100-200 записей), которые в последствии будут интегрированы в подсистему ИАД.

2.9. Архитектура системы

В качестве основного языка программирования в SKLIFF используется C# на платформе .NET. Для взаимодействия автоматизированных рабочих мест пользователей (АРМ) с СУБД Microsoft SQL Server 2000 использована библиотека классов ADO.NET. Обмен информацией с подсистемой анализа организован на основе стандарта PMML (Predictive Modeling Markup Language) [14]. Это позволяет в перспективе интегрировать в систему алгоритмы, написанные на различных языках программирования, например, адаптированные open-source реализации, а при желании обеспечить взаимодействие с другими аналитическими системами (рис. 5).

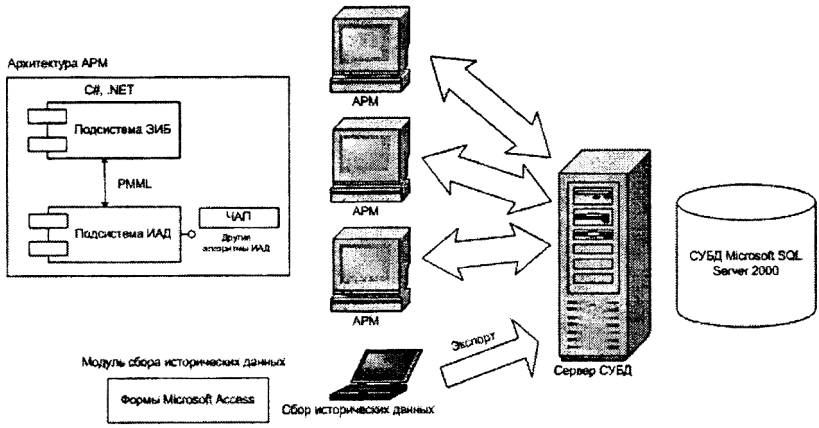


Рисунок 0. Архитектура медицинской информационно-аналитической системы SKLIFF

3. Заключение

В настоящей работе были наработаны и реализованы методы, позволяющие создавать комплексные медицинские информационно-аналитические системы, имеющие широкий круг применения и обладающие мощными и расширяемыми возможностями анализа данных. Предложенная архитектура обеспечивает возможность настройки системы на потоки данных медицинской организации уже в

процессе эксплуатации, позволяет расширять набор алгоритмов, используемых при анализе данных, накопленных в историях болезни.

На текущей момент полностью завершены работы по созданию пилотной версии системы, реализующей функциональность подсистемы ведения ЭИБ и начаты работы по реализации 2-й версии системы, полностью построенной на базе XML и предоставляющей возможности ИАД.

В качестве дальнейших перспектив развития работы следует рассматривать прежде всего развитие возможностей системы по ИАД: внедрение в систему новых алгоритмов и апробацию их работы на реальных данных. Кроме этого планируется поддержка подсистемой ЭИБ условных атрибутов событий (для тех случаев, когда наличие некоторого атрибута в форме события зависит от значения другого атрибута, например, атрибут «объем перелитой плазмы» события «протокол операции» имеет смысл только в случае, если атрибут «переливание плазмы» того же события обладает положительным значением) и динамическая (с возможностью самостоятельного создания категорий) категоризация событий вместо статического разбиения на группы: мониторинги, операции, назначения. В системе также будет реализована возможность визуализации динамики изменения значений атрибутов событий и возможность построения двумерных графиков зависимостей значений пар атрибутов.

Литература

[1] Обзор IT-стандартов в области медицины: N. Wirs «Overview of IT-Standards in Healthcare» http://www.healthcare.siemens.com/medroot/en/news/electro/issues/pdf/heft_1_00_e/05wirsze.pdf

[2] http://www.nema.org/dicom/ Официальный сайт DICOM

[3] http://www.hl7.org/ Официальный сайт Health Level Seven

[4] Компьютерные технологии в медицине, М., № 1, 1997 , с. 24

[5] М.А.Шифрин, Е.Е.Калинина, Е.Д.Калинин. От электронной истории болезни к единой системе документирования лечебного процесса. Информационно-аналитические системы и технологии в здравоохранении и ОМС. Труды всероссийской конференции. Красноярск, 2002

[6] http://www.statsoft.com Официальный сайт Statsoft – производителя пакета STATISTICA

[7] http://www.sklif-ltc.ru Официальный сайт Московского городского центра трансплантации печении НИИ СП им. Н.В. Склифосовского

[8] http://www.advalg.com Официальный сайт компании Advanced Algorithms
[9] S.Yamazaki, Y.Satomura. Standard Method for Describing an Electronic Patient Record Template: Application of XML to Share Domain Knowledge. Methods of Information in Medicine. F.K. Schattauer Verlagsgesellschaft mbH, 2000
[10]U.M Fayyad, G. Piatesky-Shapiro, P. Smyth and R. Uthurusmy. Advances in data mining and knowledge discovery. MIT Press, 1994
[11]R.Srikant, R.Agrawal. Mining Quantitative Association Rules in Large Relational Tables. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data
[12]Brossette, S.E., Sprague, A.P., Hardin, J.M., Waites, K.B., Jones, W.T., and Moser, S.A. Association rules and data mining in hospital infection control and public health surveillance. Journal of the American Medical Informatics Association 5, p. 373-381, 1998
[13]A.Salleb, T.Turmeaux, C. Vrain and C.Nortet. Mining Quantitative Association Rules in a Atherosclerosis Dataset. In Proceedings of the PKDD Discovery Challenge 2004, pp.98-103, Pisa, Italy, 2004
[14]http://www.dmg.org/ Официальный сайт DMG (Data Mining Group)

Аннотации

Брусенцов Н.П. Обобщение булевой алгебры // Программные системы и инструменты. Тематический сборник № 5. М.: Изд-во факультета ВМиК МГУ, 2004.

Булева алгебра классов обобщена введением понятия «нечеткий класс» – класс, включающий привходящие подклассы. В обобщенной таким образом алгебре естественно преодолены парадоксы «классической» (двухзначной) логики, в частности, выразимо отношение содержательного следования, адекватно алгебраизуема силлогистика Аристотеля, короче, логика обретает здравый смысл.

Доложено на Ломоносовских чтениях 2004 г. на факультете ВМиК МГУ.

Библиогр.: 4 назв.

Брусенцов Н.П., Владимирова Ю.С. Булевы уравнения и логический вывод // Программные системы и инструменты. Тематический сборник № 5. М.: Изд-во факультета ВМиК МГУ, 2004.

Распространение булевой концепции логического вывода решением уравнений алгебры классов на алгебру нечетких (допускающих привходящие подклассы) классов. Обобщение совокупностного подхода к булевой алгебре путем использования нечетких совокупностей и кодирующих их троичных ДК-шкал.

Доложено на Ломоносовских чтениях 2004 г. на факультете ВМиК МГУ.

Библиогр.: 8 назв.

Гуляев А.В. Зарецкий С.В. Проблема эффективной эксплуатации IDS. // Программные системы и инструменты. Тематический сборник № , М.: Изд-во факультета ВМиК МГУ, 2004.

В настоящей статье рассматриваются системы обнаружения вторжений в сравнении с другими средствами обеспечения безопасности. В статье рассматриваются основные подходы, применяемые для решения задачи обнаружения вторжений, показываются достоинства и недостатки этих подходов. Далее дается краткий обзор недостатков IDS. Акцентируется внимание на особенностях работы систем обнаружения вторжений, которые существенно влияют на эффективность обнаружения атак. Также предлагаются пути решения задачи повышения эффективности работы IDS.

Библиогр.: 6 назв.

Королев Л.Н., Мещеряков Д.К. Исследование возможностей предобработки речевых сигналов и использования генетического алгоритма в задаче нормализации речевых образов. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье описывается эксперимент по использованию генетического алгоритма в задаче нормализации и распознавания речевых образов. Рассматриваются вопросы деперсонализации речевых сигналов, выбора параметров преобразования, обеспечивающего нормализацию, с помощью генетического алгоритма.

Ил.: 4

Библиогр.: 5 назв.

Тихонов А.В. Среда визуализации функционирования распределенных программ. // Программные системы и инструменты. Тематический сборник № 5. М.: Изд-во факультета ВМиК МГУ, 2004.

Статья описывает настраиваемую среду визуализации, используемую для визуализации процесса выполнения распределенных программ. В статье дается архитектура среды визуализации и принципы, положенные в основу её функционирования.

Описываются алгоритмы, используемые для борьбы со сложностью.

Ил.: 4

Библиограф.: 5 назв.

Машечкин И.В., Попов И.С., Смирнов А.А. Автоматизированная библиотека контрольных точек для кластерных вычислительных систем. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Данная статья посвящена проблемам разработки и использования механизмов контрольных точек для параллельных заданий в кластерных вычислительных системах. Рассматриваются основные алгоритмы и подходы к реализации механизмов КТ. Описывается архитектура и принципы функционирования разработанной экспериментальной библиотеки контрольных точек libcheckpoint, а также её основные характеристики и текущие возможности. Приводятся результаты тестирования системы на практических вычислительных задачах и перспективы развития.

Доложено на Ломоносовских чтениях 2004г. на факультете ВМиК МГУ.

Ил.: 3.

Библиогр.:8 назв.

Воронов В.Ю., Джосан О.В., Медведев М.А., Попова Н.Н. Опыт внедрения современного математического программного обеспечения на платформе IBM Regatta // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

В данной статье проводится сравнение реализаций численных библиотек на платформе Regatta. Рассматриваются математические основы решения класса задач численных методов. Проводится выбор инструментальных средств и сравнивается качество реализации различных методов в выбранных библиотеках, а также субъективно оценивается функциональность инструментальных средств и простота их использования.

Ил.: 5, Табл.: 1.

Библиогр.: 10 назв.

Истомин Т.Е. Обзор систем параллельного программирования на основе ограниченного набора типовых алгоритмических структур // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

В настоящей статье рассматриваются один из подходов к построению высокоуровневых систем параллельного программирования и приводится обзор и сравнение некоторых из его современных реализаций. В работе рассмотрены системы (p3l, SKIE, Skel-BSP, Frame, CO2P3S и PASM.

Библиогр.: 16 назв.

Булочникова Н.М., Горицкая В.Ю., Сальников А.Н. Некоторые аспекты тестирования многопроцессорных систем.

В настоящей статье описываются разработанные тесты исследования производительности многопроцессорной системы (процессоров и коммуникационной среды). В качестве инструмента анализа предлагаются средства визуализации полученных данных, позволяющих просматривать пиковую и текущую загруженности процессоров системы, а также реализующих четырехмерную модель представления производительности межпроцессорных обменов.

Ил.: 8

Махнычев В.С. Распараллеливание сложных интеллектуальных задач, основанных на обходе дерева перебора. //

Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Применение альфа-бета алгоритмов для решения сложных интеллектуальных задач, например, таких, как игра двух соперников, с использованием мощности большого количества компьютеров, объединенных в сеть, упирается в необходимость передачи большого объема информации между ними. В работе предпринята попытка ускорить решение переборных задач в компьютерной сети за счёт применения двухуровневого альфа-бета перебора.

Для демонстрации эффективности указанной схемы реализуется программа для игры в шахматы, способная качественно анализировать шахматные партии и выигрывать у ведущих шахматистов мира.

Ил.: 6

Библиогр.: 2 назв.

Куприянов А.Е. Защищенная вероятностная маршрутизация в беспроводных сенсорных сетях. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2004.

В настоящей статье проводится анализ безопасности протоколов маршрутизации в беспроводных сенсорных сетях. Проведена классификация атак на протоколы маршрутизации, сформулированы требования, которым должен отвечать защищенный протокол маршрутизации для сенсорных сетей. Также описывается разработанный автором стек протоколов защищенной вероятностной маршрутизации, удовлетворяющий указанным требованиям. В рамках стека протоколов осуществляется аутентификация всех пересылаемых в сенсорной сети пакетов средствами симметричной криптографии, а пакеты в сенсорной сети распространяются с помощью направленных волн.

Илл.: 3

Библиогр.: 4 назв.

Куприянов А.Е. Обзор сетевых протоколов для беспроводных сенсорных сетей. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье приведен обзор проколов маршрутизации и доступа к среде передачи в беспроводных сенсорных сетях. Также рассматриваются такие вопросы, как классификация сенсорных сетей в соответствии с областью и особенностями их применения,

общесистемные характеристики сенсорных сетей, индивидуальные характеристики сенсоров, особенности функционирования сенсорных сетей.

Библиогр.: 20 назв.

Герасимов С.В. Машечкин И.В. Петровский М.И. Розинкин А.Н. Мультиагентная архитектура для Machine Learning систем фильтрации спама масштаба предприятия. // Программные системы и инструменты. Тематический сборник № 4, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье рассматривается оригинальное решение предотвращения массовых рассылок по электронной почте, основанное на использовании алгоритмов машинного обучения. Предлагается быстрый и точный алгоритм, качество классификации которого выше чем у наиболее распространенных в настоящее время методов обнаружения спама. Использование мультиагентной архитектуры позволяет легко масштабировать систему, решает проблему производительности, свойственной интеллектуальным методам, а также дает возможность строить единую корпоративную интеллектуальную систему фильтрации спама в масштабах предприятия, объединяющую различные почтовые системы.

Ил.: 3

Библиогр.: 17 назв.

Муравлев В.В. Поиск дефектов программного кода: методы и средства. // Программные системы и инструменты. Тематический сборник № 6, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье анализируются существующие подходы к автоматизации поиска дефектов проектирования в программном коде объектно-ориентированных систем; рассматриваются инструментальные средства, реализующие эти подходы. На основе проведенного анализа предлагается комбинированный подход к поиску дефектов и формулируются требования к его программной реализации.

Библиогр.: 14 назв.

Леонов М.В., Мошкин К.Б. Программный инструментарий для автоматизированного описания и идентификации сложных объектов на основе XML-технологии.// Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Рассмотрен подход к решению задач автоматизации описания и идентификации объектов с иерархической структурой с помощью

XML-описания этих объектов. Представлена программа для управления такими описаниями применительно к таксономической ботанике.

Ил. 4.
Библиогр.: 6 назв.

Глазов А.Э., Леонов М.В., Новиков М.Д. Элементы дистанционного обучения и контроля знаний на вечернем отделении факультета ВМиК. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Описаны программные средства автоматизации взаимодействия преподавателя и студентов вечернего отделения: специализированный Web-сайт и система автоматизированного тестирования, которую можно использовать для проведения контрольных работ, письменных экзаменов, анкетирования и в качестве тренажера по различным дисциплинам.

Ил. 4.
Библиогр.: 4 назв.

Полякова И.Н., Емашова О.А. Автоматическое реферирование русскоязычных текстов. Редукция предложений. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье представлен обзор разнообразных алгоритмов автоматического реферирования. Выявлены их достоинства и недостатки. Предложен собственный алгоритм, позволяющий построить реферат из частично модифицированных предложений исходного текста. Описана реализованная на основе приведенного алгоритма система. В качестве вспомогательного модуля предложен поверхностный синтаксический анализатор.

Библиогр.: 13 назв.

Бурцев А.А., Владимирова Ю.С. Средства доступа к интерфейсу Win32 API в ДССП. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Статья посвящена средствам обеспечения доступа к произвольным функциям интерфейса Win32 API из ДССП-программы.

Описываются предлагаемые средства и демонстрируются примеры их применения.

Ил.: 2

Библиогр.: 4 назв.

Мещеряков Д.К. Простой способ генерации случайных чисел с использованием примитивов ОС. // Программные системы и инструменты. Тематический сборник № 5, М.: Изд-во факультета ВМиК МГУ, 2004.

Предложен новый метод формирования случайных чисел на основе исследования значений таймера в операционной системе.

Библиогр.: 7назв.

Герасимов С.В., Машечкин И.В., Чжао А.В. Концепции и архитектура медицинской информационно-аналитической системы. // Программные системы и инструменты. Тематический сборник №5, М.: Изд-во факультета ВМиК МГУ, 2004.

В статье рассматриваются проблемы создания комплексных медицинских информационно-аналитических систем. Приводятся основные концепции и архитектура экспериментальной медицинской информационной системы, предоставляющей возможности автоматизации ведения электронной истории болезни и интеллектуального анализа медицинских данных.

Ил.: 5

Библиогр.: 14 назв.

Напечатано с готового оригинал-макета

Издательство ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 28.03.2005 г.
Формат 60x90 1/16. Усл.печ.л.12,75. Тираж 100 экз. Заказ 150.
Тел. 939-3890. Тел./Факс 939-3891.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 627 к.