онной безопасности. В частности, предложенный подход позволяет:

- минимизировать риск утери идентификатора и сопряженные с этим затраты;
- усилить систему информационной безопасности;
- выстроить систему информационной безопасности на базе единого идентификатора.

Проведенный анализ позволяет сделать заключение о пригодности того или иного биометрического идентификатора. В частности, наиболее перспективными технологиями в системе идентификации пользователя являются отпечатки пальцев и радужная оболочка глаза. Отдельного внимания требуют методы мультибиометрической идентификации.

Список литературы

1. **Beardsley, Chartles T.** Is your computer insecure? // IEEE Spectrum. 1972, Jan. P. 67—78.
2. **Соколов И. А., Будзко В. И., Синицын И. Н.** Построение информационно-телекоммуникационных систем высокой доступности // Наукоемкие технологии. 2005. № 6. Т. 6. Системы высокой доступности. 2005. № 1. Т. 1. С. 6—14.
3. **Fejfar A.** Combining Techniques for Improve Security in Automated Entry Control // Proc. of Carnahan Conf. On Crime Countermeasures. Mitre Corp. MTP-191. May 1978.
4. **Соколов А. В., Шаныгин В. Ф.** Защита информации в распределенных корпоративных сетях и системах. М.: ДМК, 2002. 656 с.
5. **Петров А. А.** Компьютерная безопасность. Криптографические методы защиты. М.: ДМК, 2000. 448 с.
6. **Программно-ашаратные** средства обеспечения информационной безопасности. Защита программ и данных: учеб. пособие для вузов / Белкин П. Ю., Михальский О. О., Першаков А. С. и др. — М.: Радио и связь, 1999. 168 с.
7. **Ushmaev O. S., Novikov S. O.** Integral criteria for large-scale multiple fingerprint solutions // Proc. of SPIE. Vol. 5404. Biometric Technology for Human Identification / Ed. by Anil K. Jain, Nalini K. Ratha. (SPIE, Bellingham, WA, 2004). P. 534—543.
8. **Синицын И. Н., Новиков С. О., Ушмаев О. С.** Развитие технологий интеграции биометрической информации // Системы и средства информатики. 2004. Вып. 14. С. 5—36.
9. **Ушмаев О. С., Синицын И. Н.** Опыт проектирования многофакторных биометрических систем // Труды VIII Международной научно-технической конференции. "Кибернетика и высокие технологии XXI века". Т. 1. С. 17—28.
10. **Ушмаев О. С., Босов А. В.** Реализация концепции многофакторной биометрической идентификации в интегрированных аналитических системах // Бизнес и безопасность в России. Январь 2008. № 49. С. 104—105.
11. **Ушмаев О. С.** Применение биометрии в аэропортах // Proc. of Biometrics TTS 2007. 22 ноября 2007 г. URL: http://www.dancom.ru/rus/AIA/Archive/RUVII_BioLinkSolutions_BiometricsInAirports.pdf
12. **Ушмаев О. С.** Реализации концепции многофакторной биометрической идентификации в правоохранительных системах. Интерполитех-2007. http://www.dancom.ru/rus/AIA/Archive/RUVI_BioLinkSolutions MultimodalBiometricsConcept.pdf
13. **Синицын И. Н., Губин А. В., Ушмаев О. С.** Метрологические и биометрические технологии и системы // История науки и техники. 2008. № 7. С. 41—44.
14. **Jain A. K., Hong L., Bolle R.** On-Line Fingerprint Verification // IEEE Trans. On Pattern Analysis and Machine Intelligence. 1997. 19 (4). P. 302—314.
15. **Jain A. K., Hong L., Pankanti S., Bolle R.** An Identity-Authentication System Using Fingerprints // Proc. of IEEE. 1997. 85 (9). P. 1365—1388.
16. **Daugman J.** The Importance of Being Random // Pattern Recognition. 2003. Vol. 36. N 2.
17. **Ушмаев О. С., Босов А. В.** Реализация концепции многофакторной биометрической идентификации в интегрированных аналитических системах // Системы высокой доступности. 2007. Т. 3. № 4. С. 13—23.
18. **Ушмаев О. С.** Сервисно-ориентированный подход к разработке мультибиометрических технологий // Информатика и ее применения. 2008. Т. 2. Вып. 3. С. 41—53.

WEB-ТЕХНОЛОГИИ

УДК 004.91

А. А. Владыкин, аспирант,
А. А. Шалыто, д-р техн. наук, проф., зав. каф.,
Санкт-Петербургский государственный университет информационных технологий, механики и оптики, e-mail: shalyto@mail.ifmo.ru

Непроцедурный текстовый язык описания автоматных обработчиков XML-документов и его применение

Предложен непроцедурный язык для описания обработчиков XML-документов, который целесообразно применять для разбора этих документов. Обработчики реализованы на основе конечных автоматов, что упрощает их структуру и понимание.

Ключевые слова: *XML, SAX-обработчики, автоматы.*

В настоящее время язык XML [1] широко используется для разнообразных целей: от хранения настроек программ до описания векторных изображений. Поэтому все больше программ требуют поддержки этого языка, а программистам все чаще приходится писать код, который считывает XML-документ и выделяет из него фрагменты, представляющие тот или иной интерес.

Нередко встречаются XML-документы размером в десятки и сотни гигабайт. Один из таких документов — полная база данных Википедии, доступная для скачивания в виде XML-файла [2].

Размер этого файла, включающего все статьи Википедии с историей их изменений, в сжатом архиватором bzip2 виде составляет 148 Гбайт.

Как прочитать такой документ и извлечь из него полезную информацию, например число перекрестных ссылок между статьями? Единственным вариантом в данном случае является использование технологии *Simple API for XML* (*SAX*) [3] и написание соответствующего обработчика, который позволит решить поставленную задачу. Известны также и другие технологии доступа к XML-данным, например *Document Object Model* (*DOM*) [3] или *Java API for XML Binding* (*JAXB*) [4], но они требуют загрузки всего документа в оперативную память. Это при таких размерах документов не представляется возможным.

Недостатком технологии SAX, затрудняющим ее использование, является сложность написания кода для реализации указанных обработчиков. В данной работе предложена технология, позволяющая значительно упростить написание обработчиков этого класса. При этом предложен непроцедурный язык описания поведения SAX-обработчика. Обработчик реализуется на основе автоматного подхода, его код генерируется автоматически.

Технология Simple API for XML

SAX — это стандартный способ разбора XML, при котором XML-разборщик (*XML-parser*) последовательно читает XML-документ и уведомляет пользовательский код (SAX-обработчик, *SAX handler*) о каждом встречающемся открывающем или закрывающем теге и о текстовом контенте между ними.

Стандартный SAX-обработчик на языке Java расширяет класс `DefaultHandler` и переопределяет следующие три метода:

```
void startElement(String uri, String localName, String qName, Attributes attrs) throws SAXException;
void endElement(String uri, String localName, String qName) throws SAXException;
void characters(char[] ch, int start, int length) throws SAXException.
```

Метод `startElement` вызывается разборщиком, когда тот обнаруживает открывающий тег, а метод `endElement` — в ответ на закрывающий тег, и, наконец, метод `characters` — для текстового контента между тегами.

SAX-обработчик как автоматизированный объект управления

Попытаемся разобраться в причинах сложности SAX-обработчиков и устранить эти причины. Этой цели можно добиться с использованием автоматного подхода [5].

SAX-обработчик получает уведомления от XML-разборщика и выполняет всю работу по преобразованию потока событий в понятные приложению команды и/или структуры данных. Он должен отслеживать текущее положение разборщика в структуре XML-документа и в зависимости от этого положения по-разному реагировать на поступающие события.

SAX-обработчик является сущностью со сложным поведением [5], так как для него характерна зависимость от предыстории. При этом SAX-обработчик можно разделить на две части:

- **управляющая часть** (автомат или система автоматов), которая отслеживает текущее положение разборщика в структуре XML-документа и выбирает соответствующие выходные воздействия на объект управления;
- **объект управления** — преобразователь потока воздействий со стороны управляющей системы в понятные приложению команды и/или структуры данных.

Наибольшую сложность обычно составляет управляющая (поведенческая) часть обработчика, поскольку отслеживание текущего положения разборщика в структуре XML-документа при традиционном подходе к программированию требует введения и учета многочисленных флагов либо других хитростей. При этом объект управления, несмотря на относительную простоту, теряется в управляющем коде.

В соответствии с парадигмой автоматного программирования, сущность со сложным поведением следует представлять в виде автоматизированного объекта управления [5]. Отделив объект управления (управляемую часть) от системы управления (управляющей части), можно значительно повысить качество кода XML-обработчика. Более того, можно автоматически генерировать управляющую часть на основе его описания на предлагаемом непроцедурном текстовом языке. Это делает написание SAX-обработчиков значительно более простым, а структуру более наглядной.

Текстовый язык описания SAX-обработчика

Рассмотрим задачу извлечения списка персоналий из таблицы, находящейся в XHTML-документе. XHTML — это HTML, соответствующий всем правилам XML. Поэтому для разбора XHTML-документа можно применить технологию SAX. Эта задача выбрана в качестве примера, поскольку на ней можно продемонстрировать сложность написания SAX-обработчика при традиционном подходе к программированию.

Алгоритм извлечения списка персоналий следующий:

- найти таблицу (`<table>`) с атрибутом `class = "header"`;
- найти таблицу без атрибута `class`;

- в таблице без атрибута пропустить первую строку (<tr>) и все строки, у которых первая ячейка имеет атрибут class = "banner";
- из оставшихся строк взять вторую и четвертую ячейки (<td>) как фамилию и имя соответственно.

В соответствии с предлагаемым подходом опишем SAX-обработчик на непроцедурном текстовом языке, специально разработанном для этой цели.

Язык этого описания состоит из следующих элементов:

- *открывающий и закрывающий теги*, являющиеся для обработчика входными воздействиями; открывающий тег может иметь ограничение на значения атрибутов (например, <td class=="banner">), выходное воздействие задается после тега в фигурных скобках (<td> {object.newPerson(); });
- *скобки, используемые для группировки тегов*; со скобками могут использоваться следующие операции:
  альтернатива (...|...);
  повторение (...)*
  опциональность (...)?

Грамматика этого языка приведена в *Приложении 1.*

Описание на предлагаемом языке SAX-обработчика, реализующего приведенный выше алгоритм, имеет вид:

```
<table class == "header"> </table>
<table class == null>
<tr> </tr>
(
   <tr>
   (
      <td class == "banner"> </td> |
      <td> {object.newPerson();} </td>
      <td> {capture();} </td> {object.setLastName
      (captured());}
      <td> </td>
      <td> {capture();} </td> {object.setFirstName
      (captured());}
   )
   </tr>
)*
</table>
```

В соответствии с автоматной парадигмой обработчик будет состоять из двух частей — объекта управления и автомата управления.

Объект управления предоставляет управляющей системе некоторый интерфейс. В этом примере — функции newPerson (), setLastName () и setFirstName (). Эти функции программист пишет *вручную.*

Автомат управления вызывает функции объекта управления в соответствии с выходными воздействиями, заданными в описании SAX-обработчика. При этом могут использоваться два вспомогательных метода, предоставляемых автоматом управления: capture () и captured (). Метод capture () инициирует захват поступающего текстового контента во временный буфер автомата. Метод captured () возвращает захваченный с предыдущего вызова capture () контент и очищает буфер.

Алгоритм построения автомата управления

Генерация управляющей части SAX-обработчика требует выделения управляющих состояний и построения переходов между ними.

Выделение состояний является достаточно простой задачей, так как состояния уже неявно выделены при составлении текстового описания XML-обработчика. Для приведенного выше примера состояния выделяются следующим образом (программа, решающая, в частности, и эту задачу, приведена в *Приложении 2*):

```
0 <table class == "header"> 2 </table> 3
<table class == null> 4
<tr> 5 </tr> 6
(
   <tr> 8
   (
      <td class == "banner"> 10 </td> |
      <td> 11 </td> 12
      <td> 13 </td> 14
      <td> 15 </td> 16
      <td> 17 </td>
   ) 9
   </tr>
)* 7
</table> 1
```

Неформально говоря, каждый открывающий или закрывающий тег переводит обработчик в новое состояние. Единственное исключение — тег в конце группы, заключенной в скобки. В этом случае новое состояние добавляется не после данного тега, а "выносится за скобки" группы, например состояние 9.

Построение переходов также не представляет большого труда. Рассмотрим переходы для некоторых состояний из примера:

- из состояния 0 существует только один переход: при получении открывающего тега table с атрибутом class, равным header, обработчик переходит в состояние 2;
- из состояния 8 возможны два перехода: в состояние 10 — при получении открывающего тега td с атрибутом class = "banner" и в состояние 11 — при получении любого тега td, который не удовлетворяет этому условию;
- из состояния 6 происходит переход в состояние 8 по открывающему тегу tr и в состояние 1 по закрывающему тегу table;
- переходы состояния 7 идентичны переходам состояния 6;
- переходы в состояния 11, 13, 14, 17 и 9 сопровождаются выходными воздействиями.

Воздействия, не предусмотренные описанием SAX-обработчика, игнорируются и не изменяют его состояние. Например, обнаружение тегов th в состоянии 5 вполне закономерно, и не должно влиять на поведение обработчика.

Программа построения управляющего автомата в целом приведена в *Приложении 2.*

Автоматическая генерация кода автомата SAX-обработчика

Имея описание управляющего автомата в виде множества состояний и переходов между ними, можно выполнить автоматическую генерацию кода автомата SAX-обработчика для произвольного языка программирования. В рамках данной работы была реализована кодогенерация автомата SAX-обработчика на языке Java.

Результатом кодогенерации является класс, который расширяет `DefaultHandler` и реализует указанные выше методы `startElement`, `endElement` и `characters`. При этом все переходы автомата по открывающим тегам размещаются в методе `startElement`, а по закрывающим — в методе `endElement`. Автомат имеет всего две внутренние переменные: текущее состояние и буфер для временного хранения текстового контента. Таким образом, расход памяти минимален.

Пример автоматически сгенерированного автомата приведен в *Приложении 3.*

В *Приложении 4* приведена структура программы SAX-обработчика.

Заключение

В связи с широким распространением языка XML задача разбора и извлечения данных из XML-документов стала рутинной. Наиболее эффективной с точки зрения использования ресурсов компьютера технологией чтения XML-документов является SAX.

В данной работе предложен непроцедурный текстовый язык описания SAX-обработчиков и система автоматической генерации кода на его основе. Входными данными для системы является простое и наглядное текстовое описание обработчика. На основе этого описания строится управляющий автомат обработчика, который затем преобразуется в код на том или ином языке программирования. На настоящий момент реализовано преобразование автомата в код на языке Java. Остальной код обработчика, соответствующий объекту управления, пишется вручную. Предложенный подход отличается одновременно простотой написания кода и эффективностью использования ресурсов компьютера.

Описанный подход был использован при разработке приложения, с помощью которого извлечены данные из тысяч HTML-страниц, содержащих несколько типов таблиц сложной структуры.

Список литературы

1. **Official website for SAX.** URL: http://www.saxproject.org/
2. **Wikipedia** Database Download. URL: http://en.wikipedia.org/wiki/Wikipedia_database
3. **W3C** Document Object Model. URL: http://www.w3.org/DOM/
4. **JAXB** Reference Implementation. URL: http://jaxb.dev.java.net/
5. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. СПб.: Питер, 2009.

Приложение 1. Грамматика описания *SAX*-обработчика

```
S :: START_TAG | END_TAG | GROUP

START_TAG :: "<" ID OR_EXPR? ">" ACTION?

END_TAG :: "<" ID ">" ACTION?

GROUP :: "(" ( START_TAG | END_TAG | GROUP )* ( "*" | "?" )? ")"

ACTION :: "{" CODE "}"

OR_EXPR :: AND_EXPR ( "||" AND_EXPR )*

AND_EXPR :: TERM ( "&&" TERM )*

TERM :: ID "==" STRING | ID "!=" STRING | ID "=~" STRING
              | ID "!~" STRING | "(" OR_EXPR ")"
```

В этом описании ID — любое корректное имя тега или атрибута; STRING — произвольный текст, заключенный в кавычки, или ключевое слово null; CODE – произвольный на целевом языке программирования.

Приложение 2. Программа построения управляющего автомата для SAX-обработчика по его описанию на предлагаемом языке

```
// Реализация используемых в программе классов для экономии места не приводится

import java.util.List;

public class StateGenerator {

    /**
     * @param input  описание обработчика в виде последовательности элементов:
     *               открывающих и закрывающих тегов и групп
     * @return сгенерированная таблица переходов
     */
    public static TransitionTable generate(List<InputElement> input) {
        TransitionTable table = new TransitionTable();
        generate(input, table, new StateSet(0), table.addState());
        return table;
    }

    private static StateSet generate(List<InputElement> input,
            TransitionTable table, StateSet initStates, int finalState) {
        StateSet states = initStates.clone();
        for (int i = 0; i < input.size(); ++i) {
            InputElement elem = input.get(i);
            int localFinalState = (i + 1 == input.size()) ?
                    finalState : table.addState();
            switch (elem.getType()) {
                case START_TAG:
                case END_TAG: {
                    for (Integer source : states.getAll()) {
                        table.addTransition(source, (Tag) elem, localFinalState);
                    }
                    states.setAll(localFinalState);
                    break;
                }
                case GROUP: {
                    Group group = (Group) elem;
                    StateSet finalStates = generate(
                            group, table, states, localFinalState);
                    switch (group.getQuantifier()) {
                        case NONE:
                        case QUESTION:
                            states = finalStates;
                            break;
                         case ASTERISK:
                             table.copyTransitions(
                                     states.getPrimary(), localFinalState);
                             states.add(finalStates, localFinalState);
                             break;
                        }
                        break;
                    }
                    default:
                        throw new IllegalArgumentException();
                }
            }
            return states;
        }

        private static StateSet generate(Group group, TransitionTable table,
                StateSet initStates, int finalState) {
            StateSet states = null;
            for (int j = 0; j < group.getAlternativeCount(); ++j) {
                StateSet finalStates = generate(
                        group.getAlternative(j), table, initStates, finalState);
                if (states == null) {
                    states = finalStates;
                } else {
                    states.add(finalStates);
                }
            }
            return states;
        }
    }
```

Приложение 3. Код автомата SAX-обработчика, сгенерированного программой из предыдущего приложения

```
public class SAXAutomaton extends org.xml.sax.helpers.DefaultHandler {

private int state;
private StringBuilder buf;
private ControlObject object;

public SAXAutomaton(ControlObject object) {
    this.object = object;
}

public void startElement(String uri, String localName,
        String qName, Attributes attrs) throws SAXException {
    switch (state) {
        case 0:
            if ("table".equals(qName) && "header".equals(attrs.getValue("class"))) {
                state = 2;
            }
            break;
        case 3:
            if ("table".equals(qName) && null == attrs.getValue("class")) {
                state = 4;
            }
            break;
        case 4:
            if ("tr".equals(qName)) {
                state = 5;
            }
            break;
        case 6:
            if ("tr".equals(qName)) {
                state = 8;
            }
            break;
        case 7:
            if ("tr".equals(qName)) {
                 state = 8;
             }
             break;
         case 8:
             if ("td".equals(qName) && "banner".equals(attrs.getValue("class"))) {
                 state = 10;
             } else
             if ("td".equals(qName)) {
                 object.newPerson();
                 state = 11;
             }
             break;
         case 12:
             if ("td".equals(qName)) {
                 capture();
                 state = 13;
             }
             break;
         case 14:
             if ("td".equals(qName)) {
                 state = 15;
             }
             break;
         case 16:
             if ("td".equals(qName)) {
                 capture();
```

```
                state = 17;
            }
            break;
    }
}

public void endElement(String uri, String localName,
        String qName) throws SAXException {
    switch (state) {
        case 2:
            if ("table".equals(qName)) {
                state = 3;
            }
            break;
        case 5:
            if ("tr".equals(qName)) {
                state = 6;
            }
            break;
        case 6:
            if ("table".equals(qName)) {
                state = 1;
            }
            break;
        case 7:
            if ("table".equals(qName)) {
                state = 1;
            }
            break;
        case 9:
            if ("tr".equals(qName)) {
                state = 7;
            }
            break;
        case 10:
            if ("td".equals(qName)) {
                state = 9;
            }
            break;
        case 11:
            if ("td".equals(qName)) {
                state = 12;
            }
            break;
        case 13:
            if ("td".equals(qName)) {
                object.setLastName(captured());
                state = 14;
            }
            break;
        case 15:
            if ("td".equals(qName)) {
                state = 16;
            }
            break;
        case 17:
            if ("td".equals(qName)) {
                object.setFirstName(captured());
                state = 9;
            }
            break;
    }
}

public void characters(char[] ch, int start, int length) {
    if (buf != null) {
        buf.append(ch, start, length);
```

```
    }
}

private void capture() {
    buf = new StringBuilder();
}

private String captured() {
    String result = buf.toString();
    buf = null;
    return result;
}
}
```

Приложение 4. *SAX*-обработчик

```
class ControlObject {
    public void newPerson() { /* реализация */ }
    public void setLastName(String name) { /* реализация */ }
    public void setFirstName(String name) { /* реализация */ }
}
class SAXHandler {
    public static void main(String[] args) throws Exception {
        SAXParser parser = SAXParserFactory.newInstance().newSAXParser();
        ControlObject object = new ControlObject();
        parser.parse(new File(args[0]), new SAXAutomaton(object));
    }
}
```

УДК 004.738

Р. Э. Асратян, канд. техн. наук, вед. науч. сотр.,
Институт проблем управления им. В. А. Трапезникова РАН, e-mail: rea@ipu.ru

Служба синхронизации процессов в сети

Описываются принципы организации новой Интернет-службы, предназначенной для синхронизации параллельных процессов в глобально-сетевой среде. Описываемая служба может рассматриваться как своего рода "сетевое обобщение" известных механизмов взаимодействия и синхронизации процессов, предоставляемых современными операционными системами (семафоры, критические секции, условные переменные и т. п.) и может быть использована в разработках распределенных информационных систем.

Ключевые слова: *распределенные системы, Интернет-службы, синхронизация процессов.*

Введение

Средства синхронизации параллельных процессов (семафоры, события, критические секции и т. п.) являются необходимым и привычным для программистов инструментом организации согласованной обработки общих данных несколькими параллельно выполняющимися процессами [1, 2]. Все современные операционные системы (ОС) включают достаточно широкий набор средств синхронизации, действующих в пределах одной машины. В последние годы потребность в подобных средствах стали испытывать и разработчики распределенных систем [3], в которых взаимодействующие процессы могут выполняться на разных сетевых узлах. К сожалению, современные сетевые технологии не предлагают каких-либо сетевых аналогов общепринятых средств синхронизации процессов, допускающих применение в многомашинной глобально-сетевой среде.

Рассмотрим, например, задачу "взаимного исключения" процессов, при доступе к общему ресурсу. В одномашинных системах эта задача эффективно решается с помощью двоичных семафоров (mutex) или критических секций, обеспечивающих строго последовательный доступ к ресурсу со стороны параллельных процессов. В локально-сетевых, моноплатформенных распределенных системах для этой цели могут быть использованы функции блокировки файлов в разделенных сетевых каталогах (системные вызовы