УДК 681.3.06

В.И. Гриценко, А.В. Анисимов, Н.Д. Пашковец, О.В. Бабак

Построение бизнес-правил для *SQL*-вложений и *JCL*-заданий

Продолжается серия публикаций о реинжиниринге программных *legacy*-систем. В данной статье изложен подход к разработке методики и алгоритмов автоматизации обработки *SQL*-вложений и построению бизнес-правил для *JCL*-заданий при проведении реинжиниринга упомянутых систем.

A series of the publications concerning the reengineering of program legacy-systems is considered. In the present article an approach to the development of a technique and the algorithms of automating the processing of SQL-enclosures and to the construction of business-rules for JCL-tasks is stated at carrying out the reengineering of the legacy-systems.

Продовжується серія публікацій про реінжиніринг програмних *legacy*-систем. В даній статті викладено підхід до розробки методики та алгоритмів автоматизації обробки *SQL*-вкладань і побудови бізнес-правил для *JCL*-завдань при реінжинірингу названих систем.

Введение. Авторы продолжают серию публикаций о различных аспектах реинжиниринга программных *legacy*-систем. В предыдущих публикациях [1, 2] были определены три основных этапа реинжиниринга программных *legacy*-систем, приведено обоснование предлагаемого подхода, описаны некоторые алгоритмы, ориентированные на автоматизацию построения бизнес-правил (БП) для первого этапа.

БП – основополагающее логическое понятие [1], на котором базируется предлагаемый подход к реинжинирингу программных *legacy*-систем, а для формального описания программных конструкций в [2] введено понятие статических и динамических векторов состояния, соответствующих наборам переменных, отвечающих свойствам и событиям проблемной области, связанных с конкретным БП.

В данной статье изложен подход к построению БП при выполнении реинжиниринга *SQL*-обращений, встроенных в программы *legacy*-систем, и описаны алгоритмы построения БП для *JCL*-заданий, инициирующих порядок выполнения программ *legacy*-систем.

Постановка задачи

Поставлена задача разработки конкретных методов и механизмов автоматического построения БП для той части *legacy*-систем, которая оперирует базами данных, с целью адекватного отражения этих процессов в новой системе. Рассмотрен вопрос описания последовательности действий, выполняемых в ходе вычислительного процесса для *legacy*-систем, с целью однозначного отображения этого процесса в новой системе.

Правильное решение рассмотренных в статье вопросов обеспечивает не только эффективный реинжиниринг, но и квалифицированное современное сопровождение *legacy*-систем с учетом особенностей их межсистемных стыковок с современными программными системами как по обмену данными, так и по управляющим воздействиям.

Обработка вложений *SQL*

Непроцедурность языка *SQL* придает ему ощутимой мощности, хотя и увеличивает ограничения, для преодоления которых *SQL* часто включают в программы, написанные на процедурном языке (*COBOL*, *PASCAL*, *FORTRAN*, *PL*/1 и др.). Поэтому при проведении реинжиниринга *legacy*-систем необходимо предусмотреть возможность обработки встречающихся вложений *SQL*.

Вложения *SQL* могут вносить коррективы непроцедурного характера в отслеживание связей наследования, в частности построения цепочек наследования. Поэтому парсер отслеживания связей наследования должен распознавать начальную фразу (*EXEC SQL*) для вложенной формы *SQL*, а также должен быть на-

строен на распознавание конечной фразы *SQL*-вложения. Соответствующее завершение для вложений *SQL* зависит от языка, для которого делается вложение: для Паскаля и *PL*/1 – точка с запятой, для КОБОЛА – слово *END-EXEC*, ФОРТРАН не имеет явного задания завершения. В других языках это зависит от реализации, и поэтому необходима предварительная настройка парсера отслеживания связей наследования на грамматику языка, допускающего вложение *SQL*. Для этого необходимо объявить секцию *SQL*, в которой указать слово-завершение для вложений *SQL*, на которое будет настраиваться парсер при обработке вложений *SQL*.

Рассмотрим обработку вложений *SQL* парсером отслеживания связей наследования. Встроенный *SQL* выполняет извлечение данных из таблиц базы данных (оператор *SELECT*) и записи данных в эти таблицы (оператор *INSERT*). Поэтому задачей парсера отслеживания связей наследования есть определение передачи результатов ввода в программу и получение вывода из программы, в которую вложены *SQL*-запросы.

Таблицы базы данных в контексте построения цепочек наследования будем рассматривать как источник входных или выходных данных для анализируемого модуля. Следовательно, парсер, войдя во вложенный *SQL* и распознав оператор *SELECT* или *FETCH* (для курсоров), заносит имена извлекаемых данных с уточнителями, какими являются имена таблиц баз данных и имена баз данных, в таблицу имен наследования модуля как имена родителей. Имена детей извлекаемых данных помещаются в таблицу имен наследования вслед за родителями. Все упомянутые имена переменных выбираются парсером из обрабатываемого *SQL*-запроса. Для распознавания имен источников данных в иерархическом представлении (имя базы данных, имя таблицы, имя переменой) потребуется дополнительный синтаксический анализ *SQL*-запроса, который выполняет специальный вспомогательный блок парсера отслеживания связей наследования. Все составляющие иерархического представления имени переменной выбираются из текста *SQL*-запроса. Но в случае курсора из оператора *FETCH* выбирается имя курсора, по которому отыскивается оператор объявления этого курсора (*DECLARE CURSOR*). Из оператора объявления курсора извлекается имя таблицы базы данных. Имя базы данных берется из оператора *CONNECT*. Аналогично, при распознавании оператора *INSERT* парсер отслеживания связей наследования заносит имена приемников данных в иерархическом (составном) представлении (имя базы данных, имя таблицы, имя переменой) в таблицу имен наследования как имена последних потомков в цепочке наследования.

Все выявленные в анализируемом модуле имена переменных из таблиц баз данных в иерархическом представлении заносятся в таблицу переменных этого модуля с отметкой как входные данные (для операторов *SELECT* и *FETCH*) или как выходные данные (для операторов *INSERT*). Такое дополнение таблицы переменных модуля позволит парсеру определения межмодульных связей учесть передачу данных между модулями через базы данных.

Далее в примере 1 показан фрагмент *COBOL*-программы с вложенными *SQL*-запросами, а в табл. 1 представлены цепочки наследования для данного примера с иллюстрацией переходов между переменными *COBOL*-программы и *SQL* вложений.

Пример 1.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG.
…
      PROCEDURE DIVISION.
…
      EXEC SQL SELECT snum, sname
          INTO :num, :salesperson
          FROM Salespeople
          WHERE snum = 1001
      END-EXEC.
      MOVE salesperson to seller.
…
      EXEC SQL DECLARE CURSOR Londonsales FOR
            SELECT *
            FROM Salespeople
            WHERE city = 'London'
      END-EXEC.
…
      EXEC SQL OPEN CURSOR Londonsales
```

```
        END-EXEC.
    …
        EXEC SQL FETCH Londonsales INTO :num, :salesperson, :loc, :comm.
        END-EXEC.
    …
        MOVE comm to switch.
    …
        EXEC SQL CLOSE CURSOR Londonsales
        END-EXEC.
    …
        EXEC SQL INSERT INTO Sales
        VALUES (:num,:seller)
        END-EXEC.
```

Таблица 1

Имена кандидатов в бизнес-термины	Имя перемен. 1	Имя перемен. 2	Имя перемен. 3	Имя перемен. 4	…	Имя перемен. *N*
Salespeople snum	*Num*	*Sales. num*				
Salespeople sname	*Sales-person*	*Seller*	*Sales. seller*			
Salespeople Londonsales comm	*Comm*	*Switch*				

Рассмотрим для вложенного *SQL* оператор *UPDATE*, который может оказаться источником репродуцирования последнего потомка в цепочке наследования.

Если в *SQL*-запросе в операторе *UPDATE* присутствует операция *SET*, присваивающая переменной некоторое значение, помещаемое в таблицу базы данных, то парсер выполняет дополнительный анализ, целью которого является установление характера выполняемого изменения значения поля в таблице базы данных. Если в случае, когда новое значение переменной получается как результат некоторых операций только над исходным значением этой же переменной, то *SET* не порождает новый бизнес-объект. Это будет тот же бизнес-объект, но с новым значением. Фрагмент программы, иллюстрирующий данный случай, показан в примере 2.

Пример 2.

```
EXEC SQL UPDATE Salespeople
    SET summa = summa + 10
    WHERE city = 'London'
END-EXEC.
```

В приведенном примере операция *SET summa = summa* + 10 не порождает новый бизнес-объект, а только присваивает обрабатываемому бизнес-объекту новое значение.

Если в получении нового значения переменной участвует хотя бы один бизнес-термин, отличный от исходного, то тогда возможны два варианта: присвоение и вычисление.

Присвоение предполагает пересылку значения одной переменной (например, из переменной главной программы) в другую (*SQL*-переменную), с которой происходит обновление поля таблицы базы данных. Следовательно, этот случай относится к наследуемым переменным (пример 3).

Пример 3.

```
EXEC SQL UPDATE Salespeople
    SET comm = :common
    WHERE city = 'London'
END-EXEC.
```

В приведенном примере *SQL*-переменная *comm*. наследует переменную *COBOL* программы common, поэтому переменная *comm*. заносится в таблицу имен наследования как потомок переменной *common*.

Вычисление значения переменной предполагает участие в *SET*-операции более одного бизнес-термина (пример 4).

Пример 4.

```
EXEC SQL UPDATE Salespeople
    SET comm = :common + :first
    WHERE city = 'London'
END-EXEC.
```

В данном случае переменная *comm*. для нашей задачи относится к классу порождаемых. Поэтому при распознавании такой операции *SET*-парсером отслеживания связей наследования, указанная переменная заносится в таблицу имен наследования как родитель цепочки наследования, а правило ее порождения из *SQL*-запроса помещается в таблицу определения порождений.

Построение БП для *JCL*-заданий

Язык управления заданиями *JCL* предназначен для описания последовательности действий, выполняемых в ходе вычислительного

процесса. Последовательность действий в языке *JCL* выражается в разбиении задания на пункты, представляющие собой, как правило, логически законченные единицы действия. Этот язык не имеет средств организации циклов и вложенностей. Допустимы только простые процессы ветвления, заключающиеся в альтернативе выполнения или обхода пункта задания, или всего задания в зависимости от результатов выполнения предыдущих его пунктов.

Задание и пункт задания включают в себя декларативные операторы, описывающие данные, и императивный оператор выполнения задания или пункта задания (*EXEC*). Основной целью описания данных является установление однозначного соответствия между данными физического уровня системы и данными логического уровня программного обеспечения, используемого в задании или пункте задания. Поэтому операторы описания данных скорее отражают техническую сторону вопроса и прямого отношения к БП не имеют.

При формировании БП для среды *JCL* следует учитывать операторы *EXEC*, ибо они, прямо или через процедуру, указывают на программу, выполняемую в пункте задания.

Всякое задание начинается с оператора *JOB*. Поставим этому оператору в соответствие некоторое условное правило активизации задания следующего вида:

IF JOB.<имя задания> *IS NOT activate*
THEN JOB.<имя задания>.

Из всех параметров, задаваемых в поле операндов оператора *JOB*, только один следует учитывать при формировании БП. Это параметр *COND*, задающий условия прекращения выполнения задания. В случае присутствия параметра *COND* (допускается до восьми таких подпараметров) правило активизации задания будет выглядеть так:

IF JOB.<имя задания> *IS NOT activate*
COND.<имя пункта задания 1>
…
COND.<имя пункта задания 8>
THEN JOB.<имя задания>,

где <имя пункта задания 1> – <имя пункта задания 8> есть имена пунктов задания, коды возврата которых анализируются на уровне задания.

Семантический смысл правила активизации задания первого вида заключается только в запуске задания, а второго – еще и в отслеживании результатов выполнения перечисленных пунктов задания, а при возникновении определенных условий – прекращение выполнения задания.

Всякий пункт задания начинается с оператора *EXEC*, которому должно соответствовать правило активизации пункта задания. Функциональным назначением этого правила есть запуск указанной программы:

IF EXEC.<имя пункта задания> *IS NOT activate*
THEN EXEC.<имя пункта задания>.

Из всех параметров, задаваемых в поле операндов оператора *EXEC*, при построении БП необходимо учитывать параметры *COND* и *PARM*.

Параметр *COND* задает условия обхода пункта задания или условия его выполнения в зависимости от того, как закончилось выполнение предыдущих пунктов (нормально или аварийно), а также в зависимости от значений кодов возврата, выработанных предыдущими пунктами. В случае наличия параметра *COND* (допускается до восьми таких подпараметров) правило активизации пункта задания выглядит так:

IF EXEC.<имя пункта задания> *IS NOT activate*
COND.<имя пункта задания 1>
…
COND.<имя пункта задания 8>
THEN EXEC.<имя пункта задания>

Через введенные правила активизации задания и пункта задания будет осуществлено связывание БП среды *JCL* и БП программ, относящихся к данному *JCL*. Эти правила носят условный характер, и, при завершении межпрограммной увязки посредством таблицы межмодульных связей, могут быть удалены, если отражают техническую сторону реализации конкретного проектного решения.

Параметр *PARM* служит для передачи программе, выполняемой в пункте задания, управляющей информации, специфичной для каждой программы. Ее состав устанавливает разработчик программы и записывается она в ви-

де строки, длина которой не превышает 100 знаков. Если строка состоит из нескольких подпараметров, разделенных запятыми, то ее заключают в апострофы или круглые скобки, которые не являются частью значения параметра (они только выделяют его). Если передаваемая с помощью *PARM* информация соотносится с бизнес-процессами, то этот факт должен быть отражен в правиле активизации, т.е. полная форма правила активизации пункта задания будет иметь следующий вид:

IF EXEC.<имя пункта задания> *IS NOT activate*
COND.<имя пункта задания 1>
…
COND.<имя пункта задания 8>
THEN EXEC.<имя пункта задания>
PARM.< управляющая информация >.

Последовательность выполнения бизнес-процессов для среды *JCL* определяется путем констатации факта установления событий, соответствующих предыдущим шагам, в условиях правил активизации пунктов задания. В табл. 2 приведен пример определения таких условий для правила активизации пункта задания. В левой колонке таблицы приведено структурное описание условий выполнения бизнес-процесса, а соответствующие им условия установления событий показаны в правой колонке.

Таблица 2

Структурное описание бизнес-процесса	Правило активизации пункта задания
IF ОТЧЕТ ПРОДАЖИ БИЛЕТОВ НЕ ОТПРАВЛЕН	*IF* <имя пункта задания> *IS NOT activate*
КАССЫ РАБОТАЛИ	*COND.*<имя пункта задания 1>
БИЛЕТЫ БЫЛИ В НАЛИЧИИ	*COND.*<имя пункта задания 2>
БЫЛИ КЛИЕНТЫ	*COND.*<имя пункта задания 3>
СОСТАВЛЕН ОТЧЕТ О ПРОДАЖЕ БИЛЕТОВ	<имя предыдущего пункта задания>
THEN ОТЧЕТ ПРОДАЖИ БИЛЕТОВ	*THEN EXEC* <имя пункта задания>
ФАКС, ОТЧЕТ ПРОДАЖИ БИЛЕТОВ ОТПРАВИТЬ	*PARM.*< управляющая информация >

Следовательно, отчет о продаже билетов будет отправлен при условии, что билеты продавались (т.е. кассы работали, были билеты, были клиенты), был составлен отчет о продаже билетов и он еще не отправлен.

В данном примере параметру *PARM* поставлено в соответствие значение ФАКС, но могли бы быть и другие значения (ТЕЛЕФОН, ТЕЛЕГРАФ, ПОЧТА и т.д.).

Для окончания задания необходимо ввести объект завершения, принимающий значения *STOPYES* и *STOPNO*, для успешного и аварийного завершения соответственно. Начальное состояние всегда *STOPYES*. Аварийное состояние устанавливается *COND*-условиями. Поэтому условное правило завершения задания будет заключаться в проверке состояния объекта завершения и выполнения соответствующего действия при выполнении активизации следующего задания, связанного с данным.

В рассмотренном примере условия выполнения пункта задания устанавливались, исходя из принятых разработчиками семантических соответствий (табл. 3).

Таблица 3

Условия активизации пункта задания	Семантическое соответствие условий активизации
<имя пункта задания> *IS NOT activate*	ОТЧЕТ ПРОДАЖИ БИЛЕТОВ НЕ ОТПРАВЛЕН
<имя пункта задания> *activate*	ОТЧЕТ ПРОДАЖИ БИЛЕТОВ ОТПРАВЛЕН
COND.<имя пункта задания 1>	КАССЫ РАБОТАЛИ
COND NOT.<имя пункта задания 1>	КАССЫ НЕ РАБОТАЛИ (следовательно пункт задания не выполняется, так как билеты не продавались)
COND.<имя пункта задания 2>	БИЛЕТЫ БЫЛИ В НАЛИЧИИ
COND NOT.<имя пункта задания 2>	БИЛЕТОВ НЕ БЫЛО В НАЛИЧИИ (следовательно пункт задания не выполняется, нечего было продавать)
COND.<имя пункта задания 3>	БЫЛИ КЛИЕНТЫ
COND NOT.<имя пункта задания 3>	НЕ БЫЛО КЛИЕНТОВ (следовательно пункт задания не выполняется, так как билеты не было кому продавать)
<имя предыдущего пункта задания> (предыдущее задание выполнено)	СОСТАВЛЕН ОТЧЕТ О ПРОДАЖЕ БИЛЕТОВ
NOT <имя предыдущего пункта задания> (предыдущее задание не выполнено)	НЕ СОСТАВЛЕН ОТЧЕТ О ПРОДАЖЕ БИЛЕТОВ (следовательно отчет не может быть отправлен)
PARM.< управляющая информация >	ФАКС, ОТЧЕТ ПРОДАЖИ БИЛЕТОВ ОТПРАВИТЬ

Исходя из изложенной специфики выделения бизнес-логики в среде *JCL* можно определить требования к парсеру обработки *JCL*-про-

грамм. Парсер распознает операторы *JOB* и *EXEC* с целью определения наличия в них *COND*-условий. Если *COND*-условия обнаружены, то их заносят в таблицу условий выполнения заданий, где фиксируется имя задания, имя пункта задания (имя программного модуля).

В примере 5 приведено задание *PROB*1, в котором поставлено условие, предписывающее прекратить выполнение задания, если код возврата любого его пункта больше восьми.

Пример 5.

```
//PROB1 JOB MSGLEVEL=1, COND=(8,LT)
…
//ST1 EXEC PGM=PROG
…
//
```

Далее в примере 6 приведено задание *PROB*2, состоящее из трех пунктов: *ST*1, *ST*2, *ST*3. В этом примере заданы условия выполнения пунктов задания. Так, пункт задания *ST*2 не будет выполняться (будет обойден), если код возврата, выработанный в пункте с именем *ST*1 будет больше семи, а пункт задания *ST*3 не будет выполняться, если код возврата, выработанный в пункте с именем *ST*1 или в пункте с именем *ST*2, будет больше семи.

Пример 6.

```
//PROB2 JOB MSGLEVEL=1
…
//ST1 EXEC PGM=PROG1
…
//ST2 EXEC PGM=PROG2, COND=(8,LE, ST1),
                                  PARM='12-25-99'
…
//ST3 EXEC PGM=PROG3, COND=((8,LE, ST1),
                                  (8,LE, ST2)),
//              PARM='12-25-01, Tuesday, night'
…
//
```

Структура строки таблицы условий выполнения заданий показана в табл. 4, где представлено ее заполнение для примеров 5 и 6.

Таблица 4

Имя задания	Имя пункта задания	*COND*1	*COND*2	*COND*3	…	*COND*8
*PROB*1		8, *LT*				
*PROB*2	*ST*2 (*PROG*2)	8, *LE*, *PROG*1				
*PROB*2	*ST*3 (*PROG*3)	8, *LE*, *PROG*1	8, *LE*, *PROG*2			

Построенная парсером таблица условий выполнения заданий окажет помощь аналитику в определении характера условий выполнения заданий и его пунктов (программных модулей). Характер этих условий определяется из установленных разработчиками семантических соответствий, подобных приведенным в табл. 3. Если какое-либо задание не может быть выполнено, поскольку не было успешно выполнено какое-то предыдущее задание, то аналитик, исходя из функционального назначения программных модулей пунктов задания, участвующих в условиях выполнения, устанавливает, имеет ли это условие бизнес-логический характер (например, билеты не продавались, поэтому отчет по продаже билетов не составляется) или технический характер (например, шаги редактирования и выполнения программы не выполнять, так как шаг трансляции выполнился неуспешно). В случае бизнес-логического характера условий выполнения задания (пункта задания) пишется БП активизации этого задания (пункта задания), которое в дальнейшем трансформируется в БП, связывающее бизнес-логику двух или более модулей системы в единый комплекс БП.

Одновременно с распознаванием *COND*-условий парсер обработки *JCL*-программ отслеживает наличие в ней *PARM*-лексем. В случае обнаружения *PARM*-лексемы подпараметрам строки управляющей информации присваиваются составные имена, состоящие из имени задания, имени пункта задания и порядкового номера подпараметра в строке, разделенные точками. Если строка управляющей информации состоит только из одного подпараметра, то порядковый номер подпараметра не указывается. Эти имена заносятся в ТПП модуля пункта задания как кандидаты в бизнес термины. При этом в качестве имени вызывающего модуля в строку таблицы передаваемых параметров будет помещено имя *JCL*, указывающее, что параметр поступил из *JCL*-программы.

Заполнение строк таблицы передаваемых параметров для примера 6 приведено в табл. 5.

Здесь *PROG*2 и *PROG*3 – имена программных модулей, выполняемых в пунктах задания.

Строка управляющей информации первого примера содержит один параметр, а второго – три параметра.

Таблица 5

Имя вызывающего модуля	Имя вызываемого модуля	Парам. 1	Парам. 2	Парам. 3	...	Парам.*N*
JCL	*PROG*2	*PROB*2. *ST*2				
JCL	*PROG*3	*PROB*2. *ST*3.1	*PROB*. *ST*3.2	*PROB*. *ST*3.3		

Кроме того, значения подпараметров строки управляющей информации заносятся в таблицу управляющей информации под присвоенными именами (табл. 6 для примера 6).

Таблица 6

Имя подпараметра	Значение подпараметра
PROB. *ST*2	12-25-99
PROB. *ST*3.1	12-25-01
PROB. *ST*3.2	*Tuesday*
PROB. *ST*3.3	*night*

Дальнейшее отслеживание продвижения параметров осуществляется через таблицу имен наследования для программ пункта задания с занесением в таблицу межмодульных связей. Аналитик, пользуясь составленной парсером таблицей управляющей информации, при анализе соответствующей *JCL*-программы устанавливает характер управляющей информации, т.е. является ли она отражением бизнес-логики процесса или управляет технической реализацией вычислительного процесса, и вносит соответствующие коррективы в таблицу межмодульных связей.

Алгоритм хронологизации дерева вызова модулей *legacy*-систем

Используемый алгоритм построения дерева вызовов модулей системы [2] строит дерево по факту взаимодействия модулей, но не отражает порядка выполнения модулей при функционировании системы. Порядок и условия выполнения модулей определяется с помощью языка управления заданиями *JCL*. С целью облегчения работы аналитика строки в таблице отображения дерева вызовов располагают в порядке выполнения модулей, отвечающим пунктам задания. Поэтому выполняется дополнительный анализ языка управления заданиями *JCL* для распознавания имен программных модулей в операторах пунктов задания (*EXEC*). Последние нумеруются порядковыми номерами, а в таблице дерева вызова программ распознанное имя программного модуля переставляется в строку, номер которой соответствует номеру пункта задания. Модифицированная таблица дерева вызова программ в сочетании с таблицей условий выполнения заданий будут отражать порядок и условия выполнения модулей системы, следовательно, представят аналитику общую картину функционирования анализируемой программной системы.

Имя программного модуля в операторе пункта задания указывается в единственном позиционном параметре, который в поле операндов всегда стоит на первом месте и который задается в одной из следующих четырех форм:

PGM = имя программы
PGM = ссылка
PROC = имя процедуры
имя процедуры.

В форме «*PGM* = имя программы» позиционный параметр указывает, что нужно выполнить программу, которая находится в постоянной (системной или личной) библиотеке.

В форме «*PGM* = ссылка» ссылка указывает на оператор описания данных, в котором описан библиотечный набор данных, содержащий вызываемую программу.

Формы «*PROC* = имя процедуры» и «имя процедуры» эквивалентны. Лексема *PROC* не обязательна, и эти формы указывают на выполнение процедуры, представляющей собой именованную последовательность управляющих операторов языка управления заданиями *JCL*, которую можно вызвать по имени и использовать в пункте задания.

Заключение. В статье описана разработка методики и алгоритмов автоматизации обработки *SQL*-вложений при проведении реинжиниринга программных *legacy*-систем.

Получены следующие результаты:

- разработан механизм отслеживания связей наследования, в частности построения цепочек наследования для вложенной формы *SQL* с целью общесистемного расширения таблиц наследования модулей;

• предложена методика настройки парсера отслеживания связей наследования на распознавание начальной и конечной фраз *SQL*-вложения для различных языков программирования;

• намечены пути автоматизации предлагаемой методики путем соответствующих дополнений, вносимых в БП при их поэтапном построении.

Кроме того, в данной статье рассмотрена разработка методики и алгоритмов автоматизации построения БП для *JCL*-заданий при проведении реинжиниринга *legasy*-систем:

• предложена методика разбора *JCL*-заданий, обработка значений параметров, задаваемых в операторах шагов задания и задания в целом, с целью определения схемы функционирования *legacy*-системы в зависимости от возникающих текущих ситуаций;

• разработаны типовые алгоритмы хронологизации вызова модулей *legacy*-системы, определения межмодульных связей и отслеживания связей наследования между переменными в масштабах шагов задания и всего задания;

• определены требования и правила построения парсера обработки *JCL*-программ, для автоматизации построения БП на рассматриваемом этапе проведении реинжиниринга программных *legacy*-систем.

1. *Реализация* реинжиниринга программных *legacy*-систем / А.В. Анисимов, В.В. Белодед, Н.Д. Пашковец и др. // УСиМ. – № 6. – 2008.
2. *Автоматизация* построения бизнес-правил в процессе реинжиниринга программных *legacy*-систем на этапе анализа их функциональных структур / В.И. Гриценко, А.В. Анисимов, В.В. Белодед и др. // УСиМ. – № 3. – С. 56–64.

Поступила 03.06.2008

Тел. для справок: (044) 259-0690, 259-0427, 526-4187 (Киев)
npashkovets@online.ua, *dep*175*@irtc.org.ua*

Окончание статьи Н.И. Ильина

10. *Интеграция* региональных систем спутникового мониторинга на основе стандартных картографических интерфейсов / Е.А. Лупян, В.П. Саворский, А.Н. Прошин и др. // Пятая Юбилейная Открытая Всерос. конф. «Современные проблемы дистанционного зондирования Земли из космоса». Москва 12–16 ноября 2007 года. – М: ИКИ РАН, 2007. – С. 1–4.
11. *GEON*: Assembling Maps on Demand From Heterogeneous Grid Sources / I. Zaslavsky, A. Memon // ESRI User Conf. – San Diego, Calif. – 2004. – P. 7–13.
12. *Lin K.*, *Ludäscher B.* A System for Semantic Integration of Geologic Maps via Ontologies // Ibid. – P. 1–10.
13. *Mapping* on the Grid: From Spatial Web Services to Mobile Clients / I. Zaslavsky, A. Memon, P. Velikhov et. al. // ICA UpiMap 2004. – Tokyo, Japan. – 2004. – P. 1–10.
14. *Шелестов А.Ю.*, *Корбаков М.Б.*, *Лобунець А.Г.* Реалізація Grid-інфраструктури для розв'язання задач обробки супутникових даних // Проблеми програмування. – 2006. – № 2–3. – С. 94–101.
15. *Shelestov A.*, *Kravchenko O.*, *Ilin M* Distributed visualization systems in remote sensing data processing GRID // Information Technologies and Knowledge. – 2008. – **2**.
16. *Online* Querying of Heterogeneous Distributed Spatial Data on a Grid / I. Zaslavsky, A. Memon, M. Petropoulos et. al. // Proc. of the 3rd Intern. Symp. On Digital Earth. 21–25 Sept. 2003. – P. 813–823.
17. *Шелестов А.Ю.*, *Куссуль Н.Н.*, *Скакун С.В.* Grid-технологии в системах мониторинга на основе спутниковых данных // Проблемы управления и информатики. – 2006. – № 1–2. – С. 259–270.
18. *Модель* GFS. – http://www.emc.ncep.noaa.gov/modelinfo/index.html
19. *WRFSI*. – http://wrfsi.noaa.gov/
20. *Сервисы* экологического мониторинга ИКИ НАНУ и НКАУ. – http://inform.ikd.kiev.ua/index.php?path=/ua/services
21. *Нейросетевой* метод мониторинга затопленных территорий с использованием радиолокационных спутниковых данных / Н.Н. Куссуль, Е.А. Лупян, А.Ю. Шелестов и др. // Исследование Земли из космоса. – 2008. – № 4. – С. 29–36.
22. *Определение* затопленных территорий на основе интеграции разнородных данных / Н.Н. Куссуль, Е.А. Лупян, А.Ю. Шелестов и др. // Проблемы управления и информатики. – 2007. – № 6. – С. 117–126.
23. *Інформаційний* сервіс оцінювання видового різноманіття рослинного і тваринного світу причорноморського регіону України в контексті розвитку українського сегмента системи GEOSS / Н.Н. Куссуль, М.О. Попов, А.Ю. Шелестов та ін. // Наука та інновації. – 2007. – № 6. – С. 13–25.
24. *OpenGIS* Web Map Service Implementation Specification. – http://www.opengeospatial.org/standards/wms
25. *OpenGIS* Web Coverage Service Implementation Specification. – http://www.opengeospatial.org/standards/wcs
26. *Apache* Jmeter. – http://jakarta.apache.org/jmeter/
27. *Менаске Д.*, *Алмейдо В.* Производительность Web-служб // Анализ, оценка и планирование. – М.: ДиаСофт, 2003. – 480 с.

Поступила 12.12.2008

Тел. для справок: (044) 526-2553 (Киев)