УДК 681.3

М.Ю. КРУКОВСКИЙ

ЯЗЫК ОБРАБОТКИ ГРАФОВ НА БАЗЕ JAVA

Abstract: *This paper describes a language for definition of workflow processes. The graph model was used as a base for the language. The language was used for creation of the system of designing and execution processes of docflow.*
Key words: *docflow, workflow, graph model, JAVA.*

Анотація: *У статті розглянуто мову, яка дає можливість описувати процеси композитного документообігу. Основою для мови була використана графова модель документообігу. Мова використовувалась для створення системи проектування та виконання процесів документообігу.*
Ключові слова: *електронний документообіг, процесне керування, графова модель, JAVA.*

Аннотация: *В статье рассмотрен язык, позволяющий описывать процессы композитного документооборота. Основой для языка послужила графовая модель документооборота. Язык использован для создания системы проектирования и исполнения процессов документооборота.*
Ключевые слова: *электронный документооборот, процессное управление, графовая модель, JAVA.*

1. Введение

Формальный синтаксис и неформальная семантика определяют основные свойства существующих языков программирования. Языки и системы программирования подчинены общим законам эволюции [1]. Эволюция языков программирования прошла через парадигмы, которые в момент внедрения считались глубоко продуманными и устойчивыми. Такие парадигмы языков, как процедурные, модульные, объектно-ориентированные, в свое время считались эффективным средством при решении прикладных задач. На сегодняшний день стало очевидно, что значимым фактором является не только синтаксис или форма отображения грамматик, но и прикладное значение языка.

В рамках настоящей статьи будет рассмотрено расширение языка JAVA, которое позволяет оперировать графами на уровне языковых конструкций. Автор пришел к необходимости данной разработки в процессе работы над реализацией системы композитного документооборота [2], основой которой выступает графовая модель [3]. Разработанное расширение распространяется с открытым кодом и может быть использовано для решения прикладных задач, оперирующих аппаратом теории графов.

2. Постановка проблемы

Для решения задачи предполагается использовать язык JAVA таким образом, чтобы с помощью его расширения можно было бы описывать и решать задачи документооборота, используя естественные для документооборота понятия и определения. В качестве основной модели предполагается использовать графовую модель, введенную автором настоящей статьи в работе [3]. Таким образом, задача создания языка документооборота сводится к задаче расширения JAVA возможностями работы с графами и наполнения этого языка семантикой документооборота.

Теория графов сегодня является очень важным и полезным аппаратом дискретной математики. Она широко применяется при решении как теоретических вопросов, так и практических инженерных задач. Особенное применение теория графов нашла при решении таких задач, как

автоматизированный контроль, сетевое планирование и проектирование интегральных схем. Кроме этих задач, очень широко графы применяются при создании моделей различного взаимодействия. Интересным является факт, что графы используются не только в перечисленных, достаточно детерминированных задачах, но и в гуманитарных науках, таких как эпидемиология и лингвистика [4].

Необходимо отметить, что предлагаемый язык программирования, позволяющий обрабатывать графы, будет полезным многим программистам и востребован для решения широкого круга самых разнообразных задач. Основным достоинством предлагаемого языка является то, что он позволяет программисту оперировать графами, используя знакомый и привычный инструментарий. Основой для такого прикладного языка, реализующего работу с графами, целесообразно выбрать уже существующий, широко используемый для общих прикладных задач язык программирования. На сегодняшний день такой платформой является доминирующий, практически без альтернативы, при разработке локальных и распределенных приложений язык JAVA. Помимо этого, язык JAVA хорошо приспособлен для решения задач сетевого взаимодействия.

2.1. Состояние проблемы

Работы по разработке языков программирования, оперирующих понятиями теории графов, ведутся с семидесятых годов прошлого столетия. Каждый из полученных языков использовался при решении прикладных задач и являлся расширением языка программирования, наиболее широко используемого на то время.

Одним из первых появился язык GEA (Graph Extended Algol), который был разработан в 1970 р. на базе системы Univac 1108 и использовался в течение десятилетия в университетских вычислительных центрах [5]. Основным достоинством языка являлось то, что при своей простоте реализации он позволял решать задачи обработки графов, используя базовые навыки программирования. Существенным недостатком этой разработки было непродуманное прикладное применение, что обусловило узкий круг использования. Широко использовался для прикладных задач язык GRASPE, построенный на базе библиотек языка Lisp [6]. Название языка Lisp (Лисп) происходит от list processing (обработка списков). Он широко используется в задачах символьной обработки, искусственного интеллекта, в математической лингвистике и др. Помимо этого, язык Lisp может быть использован для построения графиков и задания чертежей. Но так как LISP оперирует списочными структурами, то его реализация позволила не только функционально оперировать графами, но и способствовала их визуализации [7].

Впоследствии предпринимались попытки создания универсального языка, который заложил бы долгосрочную базу под будущие языки обработки графов. Один из таких языков – GXL (Graph Transformation Languge), построенный на базе существовавшего на тот момент математического языка обработки деревьев TXL (Tree transformation language) [8]. Язык был хорошо проработан с математической точки зрения, что, безусловно, обеспечивало самые широкие возможности для обработки графов. В то же время недостаточно была проработана его стыковка с языками программирования, что сделало этот язык известным только в узком кругу специалистов. Другое

семейство языков, GBL (Graph Based Language), построено в виде набора семантических определений и правил языковых цепочек с применением аппарата теории формальных языков [9]. Такой подход обеспечил достаточную общность описаний. Но вследствие недостаточной практической пользы применения, конкретных программных реализаций, основанных на этом языке, он остался невостребованным.

Таким образом, задача создания расширения языка программирования, оперирующего с графами, имеет достаточно проработанную и апробированную базу. В то же время реализаций такого языка на базе самого распространенного сейчас языка JAVA на данный момент автору неизвестно. Наиболее близкая к рассматриваемой в статье задаче – это разработанный на базе JAVA язык для иерархического моделирования и воспроизведения систем HiMASS-j (Hierarchical Modeling And Simulation System – JAVA) [10].

2.2. Выбор инструментария

Для решения поставленной выше задачи целесообразно использовать модификацию языка JAVA для реализации сложных приложений распределенных предприятий J2EE (JAVA 2 Enterprise Edition). Следует сказать, что язык J2EE делает упор не на библиотеки, а на набор связанных спецификаций и рекомендаций, которые собраны вместе для построения многоуровневых кроссплатформенных приложений. В данном контексте под спецификациями понимаются стандартизованные данные и методы их обработки, которые включены в платформу. При этом рекомендации представляют собой примеры реализаций по определенным предметным областям в соответствии со спецификой и особенностями применения.

Выбор языка J2EE в качестве исходной базы связан с тем, что именно он и его платформа рассматриваются разработчиками JAVA как наиболее перспективная среда проектирования и разработки распределенных JAVA–приложений. Данная среда была специально модернизирована при поддержке современных требований для распределенных приложений.

3. Описание системы

Как уже отмечалось, язык GJE сделан в виде расширения к существующему и широко используемому языку JAVA. Автором предполагается, что сделанные на этом языке приложения можно будет использовать во всех трех существующих реализациях конечных приложений на JAVA, а именно – локальных приложениях Application, удаленно исполняемых приложениях Applet и серверно-тиражируемых приложениях Servlet.

Технически язык GJE представляет собой внешнее расширение пакетов JAVA для решения задач документооборота и описан как package javax. workflow. Это дает возможность разработчиками подключать пакет и использовать его для решения задач документооборота.

Язык GJE позволяет строить модели документооборота, которые основаны на аппарате теории графов. Автором настоящей статьи в работе [3] введена графовая модель документооборота. Поэтому при описании классов будет даваться не только общее назначение

методов и данных с точки зрения графа как математического понятия, но и применение содержания классов, введенное в модели документооборота.

3.1. Описание интерфейсов языка GJE

Ниже приведено описание интерфейсов классов, являющихся основой языка GJE. Эти классы имеют тип “public”, поэтому они составляют основу построения конструкций документооборота на языке JAVA. Выбранная открытость реализации внешнего пакета обеспечивает возможность внесения модификаций в язык с учетом особенностей конкретных реализаций документооборота.

3.1.1. Класс Node

Класс Node представляет описание единицы нижнего уровня графа – вершины графа. Вершина графа является начальным элементом построения структуры, поэтому она не содержит методов, а содержит только значение, свойственное именно данной конкретной вершине. Как известно, совокупность вершин и их связи определяет граф.

В документообороте множеству вершин графа соответствует множество состояний документов, используемых в моделируемом документообороте. Каждая отдельная вершина соответствует отдельному состоянию документа, выделение которого считается целесообразным при дискретизации процессов документооборота. Семантическими данными этого класса является содержательная часть документа. Такими данными могут быть текст, звук, видео и другие данные, которые могут быть представлены в рамках используемых операционных систем и средств разработки.

При реализации документооборота, исходя из свойств языка JAVA, класс может быть расширен дополнительными методами обработки или данными. Такими данными является информация, которая не была известна на момент проектирования системы, а актуализировалась во время ее внедрения. Это дает возможность наращивать систему, при этом не нарушая основных правил.

Ниже приведен текст интерфейса класса Node.

```
package javax.workflow;
public interface Node
{
        Object getValue();
        void setValue(Object value) throws InvalidOperation;
}
```

3.1.2. Класс Edge

Класс Edge используется для описывания ребер графа. Ребро графа является базовым элементом аппарата теории графов и характеризуется тем, что соединяет одну или более вершин. Ребро может быть ненаправленным, то есть просто выступать элементом связности, упорядочивающим

отношения между вершинами. Направленное ребро, кроме установления факта связности, еще и определяет последовательность в иерархии, то есть указывает на причинно-следственную связь между вершинами.

В применении к графовой модели документооборота, введенной автором в работе [3], вершина графа ассоциируется с действием, которое производит участник документооборота над документом. Если говорить более строго в терминах введенной модели, то ребро графа, описывающего модель документооборота, является действием, произведение которого вызывает смену состояния документа с начального на промежуточное либо конечное. Соответственно входящей вершиной графа является входящее состояние документа. После произведения действия, установленного на ребре графа, документ принимает состояние, соответствующее исходящей вершине графовой модели.

Класс Edge содержит методы getInPoint и getOutPoint, которые используются для получения входящих и исходящих вершин соответственно. Метод getDirection получает данные, которые соответствуют направленности ребра. Метод getDirection имеет тип Object, где ребру могут соответствовать не только указания направленности, но и различные весовые характеристики. Метод setDirection используется для принудительного установления свойств направленности.

Методы getValue и setValue предназначены для получения и установления дополнительных свойств ребра. В применении к задачам документооборота это означает возможность введения дополнительной информации, свойственной действию. В частности, такой информацией является информация об исполнителях документооборота, которые могут либо должны производить это действие.

Ниже приведен текст интерфейса класса Edge.

```
package javax.workflow;
public interface Edge
{
        Node getInPoint();
        Node getOutPoint();
        Object getDirection();
        void setDirection(Object direction) throws InvalidOperation;
        Object getValue();
        void setValue(Object value) throws InvalidOperation;
}
```

3.1.3. Класс Graph

Класс Graph является классом для классов, который объединяет функциональность классов Node и Edge. В данном классе реализовано классическое представление графа в виде совокупности вершин и ребер, соединяющих некоторые из этих вершин. Класс позволяет хранить произвольное количество вершин, имеющих определенное семантическое значение. Кроме того, класс обеспечивает возможность установления и хранения произвольного количества связей между заданными вершинами.

В модели документооборота, реализованной на графах, этот класс выполняет функцию депозитария бизнес-процессов. В данном классе обеспечено хранение и управление основными данными, составляющими документооборот, а именно – функции участников документооборота, управление действиями участников и документами. Описанные с помощью этого класса процессы являются обобщенными копиями реальных процессов, происходящих в организации. При установленной общности возможно использование объектно-ориентированного наследования. Из депозитария берется копия нужного класса, по макету которого создается реализация, представляющая собой активный процесс документооборота.

Метод createNode предназначен для создания множества вершин графа, то есть для создания множества состояний документов, используемых в процессе, а метод createEdge используется для создания множества ребер графа документооборота. В применении к модели документооборота это означает множество действий, которые производятся участниками для изменения состояний документов. Методы deleteNode и deleteEdge используются при удалении вершины и ребра соответственно. Методы getNodes и getEdges используются для создания упорядоченных коллекций, являющихся хранилищем для множества вершин и множества ребер.

Методы getName и setName применяются для хранения специфической информации, которая используется для индивидуального обозначения каждого процесса. Эти методы были использованы в связи с тем, что часто возникает ситуация, когда создается много очень похожих процессов с движением многих подобных документов. В таких случаях применяется индивидуальное маркирование процессов, которое свойственно процессу в пределах его жизненного цикла.

Ниже приведен текст интерфейса класса Graph.

```
package javax.workflow;
public interface Graph
{
        Collection getNodes();
        Collection getEdges();
        Node createNode(Object value);
        Edge createEdge(Node in, Node out, Object direction, Object value);
        void deleteNode(Node node) throws InvalidOperation;
        void deleteEdge(Edge edge) throws InvalidOperation;
        String getName();
        void setName(String name);
}
```

3.1.4. Класс HyperGraph

Класс HyperGraph является высшим из классов в иерархии языка GJE. Этот класс объединяет в себе все вышеописанные классы и действия над ними. Действия над классами реализованы в соответствии с алгеброй документооборота, предложенной автором в работе [2]. Кроме алгебры

документооборота, в методах этого класса реализованы методы, обеспечивающие создание и управление коллекцией графов. Коллекция графов используется для депозитария, в котором хранятся графы.

Гиперграфы – это совокупность графов, объединенных по определенным свойствам. Гиперграфы используются для представления совокупности графов в виде единого целого без потери свойств и характеристик, присущих графам, входящим в гиперграф.

В графовой модели документооборота гиперграфом описывается совокупность процессов, составляющих документооборот предприятия либо обособленного подразделения. Гиперграф является хранилищем процессов, в котором объединены графы процессов, которые используются, использовались или могут быть использованы на предприятии.

Метод getGraphs применяется при создании и управлении депозитарием процессов, реализованных в виде коллекции графов. Тип Collection является стандартным типом языка JAVA, который используется для создания и управления большими массивами разнородных данных. Методы addGraph и deleteGraph используются для добавления и удаления графов из депозитария. Для реализации этих задач применяются стандартные средства языка JAVA по управлению коллекциями данных. В то же время в реализации языка GJE предусмотрена возможность усиления стандартных возможностей, то есть добавления собственных методов управления данными депозитария.

В классе реализована алгебра документооборота, оперирующая на графах. Реализации методов основаны на описании операций на графах, представленных в работе [10]. В настоящую реализацию языка GJE входят следующие операции: объединение, пересечение, разности множеств и декартово произведение множеств.

Метод unionGraph реализует операцию объединения графов. Объединение графов основано на операции объединения множеств. В применении к задачам документооборота эта задача используется при синтезе композитного документооборота из апробированных процессов, хранящихся в депозитарии предприятия.

Метод intersectionGraph используется для реализации операции пересечения графов. Необходимость применения этой операции в графовой модели документооборота возникает, когда требуется получить пересечение существующих процессов. При реализации систем документооборота часто получаются системы, состоящие из значительного количества процессов. В таких случаях возникает задача анализа процессов для выявления конфликтных и дефицитных исполнительских ресурсов. После применения операции пересечения к нескольким графам получается результирующий граф, являющийся графом критического пути. То есть такого пути, появление задержек на котором вызовет возникновение задержек во всем документообороте.

Метод differenceGraph реализует операцию разности на графах. В применении к модели документооборота это означает разницу двух процессов. Данная операция используется, когда возникает задача сравнения различных процессов. Обычно такая задача возникает в том случае, если надо выделить неиспользуемый фрагмент процессов. Для этого используется так называемый эталонный процесс. Во время анализа считается, что содержание эталонного процесса является наилучшим для решения рассматриваемого класса задач. Такие эталонные

процессы хранятся в депозитарии и могут являться основой документооборота при синтезе сложных процессов. После сравнения эталонного процесса с изучаемым получается разница, которую следует критически рассмотреть на предмет возможного улучшения.

Метод cartesianGraph применяется для реализации произведения графов. В задачах документооборота произведение графов используется при получении декартового произведения процессов. Например, необходимость в этом появляется в случае, если возникает задача тиражирования процессов документооборота с некоторым предопределенным параметром. Частным случаем такой задачи можно считать тиражирование с единичным вектором (скаляром) после умножения, на который процесс не изменяется. Таким образом, задача тиражирования процесса с параметром может быть представлена в виде цикла умножения матрицы процесса на матрицу-вектор, которая изменяется от скаляра до установленного значения.

Ниже приведен текст интерфейса класса HyperGraph.

```
package javax.workflow;
import java.util.Collection;
public interface HyperGraph
{
        Collection getGraphs();
        void addGraph(Graph graph)  throws InvalidOperation;
        void deleteGraph(Graph graph) throws InvalidOperation;
        Graph unionGraph(Graph graph1, Graph graph2);
        Graph intersectionGraph(Graph graph1, Graph graph2);
        Graph differenceGraph(Graph graph1, Graph graph2);
        Graph cartesianGraph(Graph graph1, Graph graph2);
        Graph createGraph(Collection nodes, Collection edges);
}
```

3.1.5. InvalidOperation

Класс InvalidOperation используется для обработки исключений. Исключения возникают при выполнении операций с депозитариями, не предусмотренными стандартными описателями, а также при некорректных операциях на графах. Этот класс можно использовать для дополнительной индивидуализации приложений, поскольку ему и передается управление в случае возникновения внештатных ситуаций.

В настоящей реализации для обработки исключений используется конструктор родового класса. Что и позволяет разработчику применять собственные методы обработки исключений. Это обеспечивает дополнительную совместимость и гибкость реализации.

Ниже приведен текст интерфейса класса InvalidOperation.

```
package javax.workflow;
public class InvalidOperation
        extends Exception
{
```

```
        public InvalidOperation(String message)
        {
                super(message);
        }
}
```

4. Выводы

В настоящей статье представлен язык обработки графов GJE на базе расширения языка JAVA, который был использован для создания системы проектирования и исполнения систем композитного документооборота. Наряду с операциями над множествами, дано описание интерфейсов для классов вершин, ребер, графовых систем и их объединение. Показана возможность языка GJE как для анализа, так и синтеза системы композитного документооборота.

Благодаря построению языка GJE как расширения языка JAVA имеется возможность обеспечить локальное и сетевое взаимодействие между процессами электронного документооборота и адаптацию систем к внутренним и внешним условиям использования.

СПИСОК ЛИТЕРАТУРЫ

1. Теслер Г.С. Новая кибернетика. – Киев: Логос, 2004. – 401с.
2. Круковский М.Ю. Концепция построения моделей композитного документооборота // Математичні машини і системи. – 2004. – № 2. – С. 149 – 163.
3. Круковский М.Ю. Графовая модель композитного документооборота // Математичні машини і системи. – 2005. – № 1. – С. 120 – 163.
4. Watts D.J. Small worlds: the dynamics of networks between order and randomness. – Princeton: Princeton university press, 1999. – 262 p.
5. Crespi-Reghizzi S., Morpurgo R. A language for treating graphs // Communications of the ACM. – 1970. – Vol. 13, Issue 5. – P. 319 – 323.
6. Хювенен Э., Сеппянен Й. Мир Лиспа: В 2 т. – М.: Мир, 1990. – Т. 1: Введение в язык Лисп и функциональное программирование. – 447 с.; Т. 2.: Методы и системы программирования. – 319 с.
7. Pratt T.W., Friedman D.P. A language extension for graph processing and its formal semantics // Communications of the ACM. – 1971. – Vol. 14, Issue 7. – P. 460 – 467.
8. Sarkar M.S. GXL: a new graph transformation language // Proc. of the 42nd annual southeast regional conference. – 2004. – P. 336 – 340.
9. Kleyn M.F., Browne J.C. A high level language for specifying graph based languages and their programming environments // Proc. of the 15th international conference on Software Engineering. – 1993. – P. 324 – 335.
10. Daum Th., Sargent R.G. A Java based system for specifying hierarchical control flow graph models // Proc. of the 29th conference on Winter simulation. – 1997. – P. 150 – 157.