14. Ильюшин Ю. В. Проектирование распределенной системы со скалярным воздействием // Науч. обозрение. 2011. № 4. С. 85–90.

15. Ильюшин Ю. В., Кравцова А. Л., Мардоян М. М. Устойчивость температурного поля распределенной системы управления // Науч. обозрение. 2012. № 2. С. 189–197.

16. Исследование устойчивости теплового поля туннельной печи конвейерного типа / Ю. В. Ильюшин, А. Л. Кравцова, М. М. Мардоян, А. В. Санкин // Науч. обозрение. 2012. № 4. С. 114–120.

17. Ильюшин Ю. В. Методика синтеза нелинейных регуляторов для распределенного объекта управления // Науч. обозрение. 2012. № 5. С. 14–17.

18. Анализ температурного поля цилиндрического объекта управления / Ю. В. Ильюшин, И. А. Кучеренко, А. Л. Ляшенко, С. Л. Морева // Науч. обозрение. 2013. № 3. С. 71–75.

Yu. V. Ilyushin
Saint-Petersburg Mining University

I. M. Novozhilov
Saint Petersburg Electrotechnical University «LETI»

MATHEMATICAL MODELS OF OBJECTS WITH DISTRIBUTED PARAMETERS WITH IMPULSE INPUT EFFECT

The peculiarity of the control theory is that it completely distracts (abstracts) from the purpose, physical nature and design features of the ACS and its constituent elements, and the study of processes and characteristics is carried out using abstract mathematical descriptions (models). The mathematical model of the ACS is a record of all those physical laws that govern changes in physical variables and phenomena in the system under study. Obtaining a mathematical model is one of the most important initial stages of research, immediately preceding the use of methods of analysis and synthesis. The article discusses the method of constructing a mathematical model of a cylindrical heating element. Three construction methods are presented: two grid methods of different dimensions and the method based on the Green function. On the basis of the experiment, we can conclude that the results obtained are identical, and as a result, there is no need to overload the mathematical apparatus with one of the methods.

System analysis, management, distributed systems, absolute stability

УДК 004.51

А. В. Леонов
Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

Алгоритм линеаризации FODA-деревьев пользовательских интерфейсов

Представлен алгоритм линеаризации деревьев пользовательских интерфейсов, построенных при помощи методики FODA, оценена его вычислительная сложность, показана эффективность алгоритма на сгенерированных данных по времени и памяти. Разработанный алгоритм позволяет представить древовидную структуру семейства пользовательских интерфейсов в виде списка линейных структур, которые могут быть использованы в существующих алгоритмах кластеризации категориальных данных для построения классов пользовательских интерфейсов. Показано, что на основе четырех типовых структур в FODA-деревьях и их однозначном разложении в линейные структуры данных можно построить алгоритм линеаризации FODA-деревьев. Каждый узел дерева представлен в виде структуры данных с набором полей, содержащим сведения по параметрам и признакам узла. Алгоритм реализован на языке Java версии 8. Проверка работоспособности алгоритма проведена при помощи фреймворка для написания тестов Junit версии 4. Для оценки времени работы алгоритма в зависимости от исходных данных был написан генератор FODA-деревьев.

Методика FODA, пользовательский интерфейс, древовидные структуры данных, алгоритмы кластеризации, сложность алгоритмов

На сегодняшний день существует большое число различных пользовательских интерфейсов (ПИ), которые оказывают огромное влияние на эффективное и качественное взаимодействие человека

с программно-аппаратными комплексами. Для обеспечения такого взаимодействия необходимо учитывать характеристики уже спроектированных ПИ на основе их классификации по определенному списку параметров (например, элементы управления системой, навигация между блоками системы, визуальный дизайн системы и т. д. [1]). Как показано в [2], ПИ могут быть представлены в виде деревьев, построенных на основе методики FODA для определения их общих черт и различий.

Разработанная методика позволяет организовать пользовательский интерфейс как дерево с различными операциями между узлами, которые представляют собой характеристики ПИ. Однако на текущий момент не существует алгоритмов кластеризации таких древовидных структур данных.

Для решения этой проблемы в статье приведено описание алгоритма преобразования FODA-деревьев ПИ в набор линейных структур (списков), которые могут быть использованы в существующих алгоритмах кластеризации категориальных данных (таких, как CLOPE [3], *k*-modes [4], ROCK [5]).

Методика FODA (Feature-Oriented Domain Analysis) [5] позволяет представить рассматриваемый ПИ в виде дерева, корнем которого является сам ПИ, а узлами – атрибуты ПИ и их значения. Узлы дерева могут быть двух видов – обязательные (закрашенный круг на атрибуте/значении) и опциональные (незакрашенный круг на атрибуте/значении). Они связываются между собой при помощи трех видов связей – простая, альтернативная и расширенная альтернативная связи.

Комбинация видов узлов и типов связей дает 4 типовые структуры в FODA-деревьях (рис. 1), которые могут быть представлены в виде линейных структур следующим образом (остальные комбинации при линеаризации сводятся к приведенным ниже четырем, поэтому не приведены).

Обязательная типовая структура представляет собой несколько узлов, связанных между собой простыми связями (линиями) с закрашенными кругами на концах. Она означает, что атрибуты дерева должны обязательно присутствовать в результате процесса линеаризации. Таким образом, обязательная структура на рис. 1 представляет собой следующий линейный вид: {UI, F1, F2}.

Опциональная типовая структура представляет собой несколько узлов, которые связаны простыми связями (линиями) с не закрашенными кругами на концах. Она означает, что такие атрибуты дерева могут либо присутствовать, либо нет в результате процесса линеаризации. Следовательно, опциональная структура на рис. 1 представляет собой уже две линейные структуры: {{UI, F1, F2}, {UI, F2}}.

Альтернативная типовая структура представляет собой две линии с закрашенными кругами на концах, соединенные между собой дугой. Она означает, что либо первый атрибут дерева, либо второй атрибут дерева попадет в результат линеаризации. Таким образом, альтернативная структура на рис. 1 представляет собой две линейные структуры: {{UI, F1}, {UI, F2}}. В данном соединении один атрибут может быть опциональным, тогда если атрибут F1 опциональный, то результат линеаризации включает 3 списка: {{UI}, {UI, F1}, {UI, F2}}. В данной структуре оба узла не могут быть опциональными, поскольку это равносильно двум простым связям с опциональными узлами.

Расширенная альтернативная типовая структура представляет собой две линии с закрашенными кругами на концах, соединенные между собой закрашенной дугой. Она означает, что-либо первый атрибут дерева, либо второй атрибут дерева, либо оба атрибута попадут в результат линеаризации. Таким образом, расширенная альтернативная связь на рис. 1 представляет собой 3 линейные структуры: {{UI, F1}, {UI, F2}, {UI, F1, F2}}. Если в данном виде связи один атрибут опционален, то результат линеаризации эквивалентен результату с двумя опциональными атрибутами.

Таким образом, на основе четырех типовых структур в FODA-деревьях и их однозначном разложении в линейные структуры данных можно построить алгоритм линеаризации FODA-деревьев.

Пусть каждый узел дерева представлен в виде структуры данных (рис. 2), содержащей следую-

Обязательная | Опциональная | Альтернативная | Расширенная альтернативная

Рис. 1

щие поля: уникальный идентификатор узла, контент узла, тип связи от его родителя, ссылку на родителя, количество альтернативных связей, список детей, а также признак обязательности узла. Уникальный идентификатор узла позволяет определить, существует ли цикл между узлами при неправильных исходных данных.

```
class FodaTreeNode {
String uuid;
    String payload;
LinkTypelinkType;
FodaTreeNode parent;
booleanisRequired = true;
intalternativesCount;
    List<FodaTreeNode> children;
}

enumLinkType {
    SIMPLE_LINK,
    ALTERNATIVE,
    EXTENDED_ALTERNATIVE
}
```

Рис. 2

Тип связи от родителя представлен в виде перечислимого типа данных, который содержит описанные выше типы связей в FODA-деревьях (рис. 2). Количество альтернативных связей позволяет оценить, что для текущего узла существует более одной пары альтернатив на одном уровне для обработки. Такая структура позволяет легко описать дерево с учетом типов связей между родителями и их потомками.

На вход алгоритма подается корень дерева, а в результате работы получается объект, который содержит в себе список из списков структур, представленных на рис. 3, поскольку одно FODA-дерево может состоять из нескольких различных списков (например, если имеется хотя бы один опциональный узел). Результирующая модель включает в себя только уникальный идентификатор и контент:

```
class LinearizedFodaTree {
    List<List<FodaNode>>
    linearizedTreeViewList;
}
public class FodaNode {
    String uuid;
    String payload;
}
```

Рис. 3

Далее представлен псевдокод алгоритма линеаризации FODA-дерева (рис. 4). Для обхода дерева используется алгоритм обхода в ширину с использованием очереди вершин. Для каждой вершины дерева при помощи процедуры findLinearStructuresToUpdate находятся списки, в которые данная вершина будет добавлена (списки определяются на основе родителя данного узла, а также типа связи и количества обработанных альтернатив для этого узла). Далее в зависимости от типа связи определяется алгоритм обработки текущей вершины.

Если связь простая и узел является обязательным, то данная вершина добавляется во все найденные списки (процедура processCurrentNode (рис. 5)). Если же вершина опциональна, то происходит копирование текущего списка для обновления и вставка в скопированный список новой вершины.

Если связь альтернативная и текущий узел является первой альтернативой, обрабатываем вершину как в случае с простой связью (при помощи процедуры processCurrentNode). Если связь альтернативная и это второй узел внутри альтернативы, то необходимо создать копию текущего списка, удалить из него первую альтернативу, добавить в него текущую и добавить скопированный список в результат.

Если связь расширенная альтернативная и текущий узел является первой альтернативой, тогда если вершина обязательна, то она добавляется в результирующий список (если вершина необязательная, это означает, что были введены некорректные данные и будет получено сообщение об ошибке). Если это второй узел внутри альтернативы, то создаются две копии текущего списка, из первого скопированного списка удаляется первая альтернатива и добавляется вторая, а также ко второму скопированному списку добавляется вторая альтернатива, далее скопированные списки добавляются к результату.

Сложность представленного алгоритма по времени в худшем случае составляет $O(n^3)$, где n – количество вершин входного дерева [6]–[8]. Такая сложность возникает в случае, если все типы связей в дереве альтернативные. Это означает, что процедура поиска линейных структур для модификации требует времени $O(n)$ (поскольку максимальное количество списков будет равно количеству элементов в дереве), а также обработка найденных структур занимает линейное время $O(n)$.

```
linearize(FodaTreeNode root) {
    nodesQueue = new Queue(root);

    while (!nodesQueue.isEmpty()) {
        currentNode = nodesQueue.poll();

        linearStructuresToUpdate = findLinearStructuresToUpdate();

        switch (linkType of currentNode) {
            case SIMPLE_LINK:
                for (linearStructure in linearStructuresToUpdate) {
                    processCurrentNode();
                }
            case ALTERNATIVE:
                if (currentNode is the first alternative) {
                    for (linearStructure in linearStructuresToUpdate) {
                        processCurrentNode();
                    }
                } else if (currentNode is the second alternative) {
                    for (linearStructure in linearStructuresToUpdate) {
                        Create a copy of current linear structure
                        Remove first alternative from copied list
                        Add second alternative to the copied list
                        Add copied list to result
                    }
                }
            case EXTENDED_ALTERNATIVE:
                if (currentNode is the first alternative) {
                    for (linearStructure in linearStructuresToUpdate) {
                        linearStructure.add(currentNode);
                    }
                } else if (currentNode is the second alternative) {
                    for (linearStructure in linearStructuresToUpdate) {
                        Create two copies of current linear structure
                        Remove first alternative from first copied list
                        Add second alternative to both copied list
                        Add two copies to result
                    }
                }
        }

        Add all children from current node to the nodesQueue
    }
}
```

Рис. 4

```
    processCurrentNode(FodaTreeNode currentNode) {
        if (currentNode.isRequired) {
            Add currentNode to currently processing linear structure
        } else {
            Make a copy of currently processing linear structure
            Add currentNode to copied list
            Add copied list to result
        }
    }
```

Рис. 5

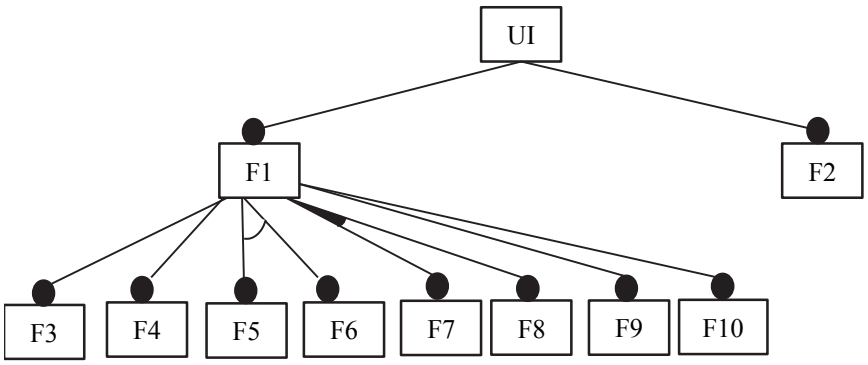

Рис. 6

С точки зрения затрат по памяти алгоритм требует дополнительного хранения множества обработанных вершин и очереди узлов для последующей обработки. Несмотря на сложность по времени, алгоритм может использоваться для решения реальных задач линеаризации FODA-деревьев ПИ за счет не очень большого количества узлов.

Алгоритм реализован на языке Java версии 8. Проверка работоспособности алгоритма производилась при помощи фреймворка для написания тестов Junit версии 4.

Для оценки времени работы алгоритма в зависимости от исходных данных был написан генератор FODA-деревьев, который позволяет создавать деревья с заданным числом узлов и равномерным распределением типов связей для них (рис. 6 – пример сгенерированного тестового FODA-дерева из 11 узлов).

Для оценки времени работы и количества затраченной памяти были сгенерированы FODA-деревья, которые содержали от 10 до 100 узлов с шагом в 10 узлов. Тесты производились на компьютере с процессором Intel Core i7 2,8 GHz и 4 Gb оперативной памяти, выделенной под Java-процесс.

По каждому количеству узлов тесты повторялись 30 раз для усреднения времени работы алгоритма, поскольку в генераторе количество узлов для каждого родителя выбиралось случайным образом (из списка от 2 до 10 узлов), что влияет на глубину результирующего дерева.

Табл. 1 содержит среднее время работы алгоритма линеаризации в зависимости от количества узлов в дереве.

Из полученных результатов можно сделать вывод, что для 100 узлов с равномерным распределением типов связей время линеаризации всего 6.7 с, что является хорошим показателем по быстродействию.

Таблица 1

Количество узлов в дереве (без учета корневого узла)	Среднее время работы, мс
10	2.50
20	4.53
30	6.06
40	7.63
50	19.4
60	47.56
70	174.06
80	367.33
90	1646.83
100	6 744.86

При этом еще одной важной характеристикой работы алгоритма является число линеаризованных списков в зависимости от количества узлов в дереве, поскольку оно влияет на используемую алгоритмом память.

В табл. 2 приведены результаты по среднему количеству линеаризованных списков в зависимости от числа узлов в дереве. Для усреднения количества списков тесты проводились 30 раз для каждой позиции.

Таблица 2

Количество узлов в дереве (без учета корневого узла)	Среднее количество линеаризованных списков
10	9
20	35
30	82
40	345
50	923
60	3 431
70	12 119
80	38 976
90	175 879
100	307 366

По результатам тестов видно, что количество линеаризованных списков растет с увеличением количества узлов в дереве и зависит от количества альтернативных и расширенных альтернативных связей в дереве, так как для этих типов

связей необходимо создавать копии списков, что отрицательно сказывается на количестве используемой памяти.

На основе полученного числа списков для узлов, а также хранимых в них объектов можно оценить используемую память.

Таким образом, в статье представлен алгоритм линеаризации FODA-деревьев ПИ, который позволит применить алгоритмы кластеризации категориальных данных к полученным спискам характеристик ПИ, приведен пример сгенерированного FODA-дерева, проведен анализ времени работы алгоритма и количества потребляемой им памяти на основе разработанного генератора FODA-деревьев.

СПИСОК ЛИТЕРАТУРЫ

1. Сергеев С. Ф., Падерно П. И., Назаренко Н. А. Введение в проектирование интеллектуальных интерфейсов: учеб. пособие. СПб.: Изд-во СПбГУ ИТМО, 2011. 108 с.

1. Назаренко Н. А., Леонов А. В., Шумская Д. Э. Методика определения профессионально важных качеств на основе особенностей пользовательского интерфейса АРМ //Изв. СПбГЭТУ «ЛЭТИ». 2017. № 7. С. 38–45.

2. Yang Y., Guan X., You J. CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data // In Proc. of SIGKDD. Эдмонтон, Канада, 2002. P. 682–687.

3. Huang Z. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values // Data Mining And Knowledge Discovery. 1998. Vol. 2, № 3. P. 283–304.

4. Guha S., Rastogi R., Shim K. ROCK: A Robust Clustering Algorithm for Categorical Attributes. // Proc. 15th Intern. Conf. on Data Engineering, Ausralia, Sydney, 1999. С. 512–521.

5. Spencer Feature-Oriented Domain Analysis (FODA) Feasibility study / C. Kang Kyo, G. Cohen Sholom, A. Hess James, W. E. Novak, A. Peterson. URL: http://www.floppybunny.org/robin/web/virtualclassroom/chap12/s4/articles/foda_1990.pdf (дата обращения 10.11.2018).

6. Алгоритмы. Построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. М.: Издательский дом «Вильямс», 2018. 1328 с.

7. Скиена С. С. Алгоритмы. Руководство по разработке. СПб.: БХВ-Петербург, 2017. 720 с.

8. Седжвик Р., Уэйн К. Алгоритмы на Java. М.: Издательский дом «Вильямс», 2016. 848 с.

A. V. Leonov
Saint Petersburg Electrotechnical University «LETI»

AN ALGORITHM OF LINEARIZING FODA TREES OF USER INTERFACES

Presents an algorithm for linearization of user interface trees built using the FODA technique, evaluates its computational complexity, shows the effectiveness of the algorithm on the generated data on time and memory. The developed algorithm allows one to present the tree structure of the user interface family in the form of a list of linear structures that can be used in existing clustering algorithms of categorical data to construct classes of user interfaces. It is shown that on the basis of four typical blocks in FODA trees and their unique decomposition into linear data structures it is possible to construct an effective algorithm for linearization of FODA trees. Each node of the tree is represented as a data structure with a set of fields containing information on the parameters and characteristics of the nodes. The algorithm is implemented in Java language version 8. Performance testing of the algorithm was conducted by using framework for writing tests Junit version 4. To estimate the time of the algorithm depending on the source data, a generator of FODA trees was written.

FODA, user interface, tree data structures, clustering algorithms, algorithm analysis