УДК 004.42

**А. С. ПРИГОЖЕВ, Д. А. НЕИЗВЕСТНЫЙ, О. С. ЛАРИОНОВА**

***Одесский национальный политехнический университет, Украина***

# АРХИТЕКТУРА СРЕДЫ ТЕСТИРОВАНИЯ НА ОСНОВЕ МОДЕЛИ ГИБРИДНЫХ РЕСУРСНЫХ СЕТЕЙ

*В статье предложена модель для тестирования программного обеспечения на основе гибридных ресурсных сетей. Определено понятие десятичного эквивалента структуры данных. Представлен метод проведения тестирования на основе гибридных ресурсных сетей, десятичного эквивалента и дифференциального исчисления. Представлена архитектура системы для автоматизированного тестирования и верификации программного обеспечения (ПО). Рассмотрены одни из основных компонентов системы: менеджер задач, визуальный редактор графов с применением силовых алгоритмов позиционирования вершин графа в визуальном редакторе и редактор спецификации. Спроектирована архитектура визуального редактора графов с применением шаблонов проектирования. Предложена реализация менеджера задач в виде RESTful веб-сервиса.*

***Ключевые слова:*** *тестирование, ресурсная сеть автоматизация, распределение задач, граф, визуализация.*

## Введение

Современная разработка программных систем базируется на основе гибких методологий разработки программного обеспечения (Agile-методологий) [1]. Основополагающими положениями данной методологии являются:

– люди и их взаимодействие важнее процессов и инструментов;

– готовый продукт важнее документации по нему;

– сотрудничество с заказчиком важнее жёстких контрактных ограничений;

– реакция на изменение важнее следования плану.

Приведенные принципы означают, прежде всего, необходимость вовлечения заказчика в процесс разработки с момента старта проекта. В этом случае, значительную роль в успехе проекта играют правильное понимание заказчиком особенностей своей предметной области и правильная интерпретация разработчиками терминологии и процессов предметной области заказчика. Отсутствие этого можно приводить к достаточно серьёзным последствиям для программного проекта. По статистике, только в США в 2010 году среди проектов разработки ПО с бюджетом не менее 15 млн. долларов: 66% превышали свой первоначальный бюджет, 33% превышали свои временные рамки, 17% имели риск упущенной выгоды, а превышение суммы указанных показателей свыше 100% свидетельствует о наличии сразу нескольких из перечисленных выше недостатков [2]. В большинстве случаев, это вызвано либо отсутствием полного понимания процессов или терминологии предметной области заказчиком либо отсутствием точного понимания процессов и терминологии предметной области разработчиками.

Ключевым этапом, на котором возможно, в процессе разработки, установить соответствие программы и её спецификации является тестирование ПО. Основными проблемами при этом являются большая разветвлённость процесса решения задачи, большое количество тестовых наборов (схемы алгоритмов, UML диаграмм, описания интерфейсов, исходных кодов программы), а также отсутствие формальных критериев, позволяющих однозначно оценить соответствие программы и спецификации.

Одним из современных направлений тестирования ПО является тестирование на основе моделей, при котором строится некоторая абстрактная модель системы, на ней проводится исследование функционирования системы, а уже потом, на основе модельных результатов пишутся тесты для реальной системы. Однако процесс такого моделирования и написание самих тестов также подвержено описанным ранее рискам, которые на этапе тестирования связаны с пониманием тестировщиком терминологии и процессов предметной области ПО.

Поэтому, актуальной задачей в таких условиях становится разработка автоматизированных систем тестирования ПО, в которой модель строится, в значительной степени, автоматически, с использованием функциональных преобразований программы и спецификации.

Целью работы является повышение точности проведения тестирования ПО, за счёт разработки автоматизированной системы тестирования ПО на основе новой модели функционирования ПО.

Для достижения поставленной цели необходимо решить следующие задачи:

– анализ современных моделей, применяемых в тестировании ПО;

- разработка модели для тестирования ПО;

– разработка архитектуры распределённой среды для тестирования ПО;

– разработка программной реализации разработанной архитектуры.

## 1. Существующие модели для автоматизированного тестирования

Системы тестирования на основе моделей базируются на применении различных математических абстрактных понятий, которые описывают определённую сторону функционирования разработанной программной системы. Вследствие этого, тесты, разрабатываемые на основе моделей также являются абстрактными и требуют детализации для конкретной программной системы. Поскольку чаще всего модель описывает какую-либо сторону функционирования готовой системы, то этот подход можно отнести к методам тестирования «чёрного ящика».

Наиболее часто применяемыми моделями в системах тестирования являются различные модификации конечных автоматов, модели Крипке, сети Петри.

Модель Крипке является простой абстрактной машиной, позволяющей описать идеи вычислительной машины без добавления особых сложностей. Модель представляется ориентированным графом, вершины которого описывают достижимые состояния системы, а ребра — переходы из состояния в состояние. Функция пометок сопоставляет каждой вершине множество свойств, которые выполняются в соответствующем состоянии. Функция пометок определяет множество логических утверждений, верных в некоторой вершине. Каждое утверждение формулируется в виде некоторого логического выражения. Это выражение, чаще всего, формулируется непосредственно тестировщиком [3]

Конечных автоматы в общем случае – пятёрка, состоящая из множества входных сигналов, множества внутренних состояний, начального состояния, множества конечных состояний и функции выходов, областью отправления которой является множество пар «состояние-входной сигнал», а областью прибытия – множество состояний автомата. Сети конечных автоматов широко применяются при верификации моделей программ. В частности, сети автоматов широко применяются при автоматизированной модификации программного кода [3]. Такой подход позволяет учитывать иерархическую структуру программного кода, а также однозначно представлять межпроцессное взаимодействие. Это позволяет автоматизировать анализ последовательностей операций процесса. Однако при этом подходе на разработчика тестов ложится необходимость установления причины такого поведения программы. Ещё одним важным недостатком является самостоятельное решение задачи построения эталонной модели функционирования разработчиком.

Сеть Петри, в общем случае представляет собой двудольный граф. Вершины одного типа не могут соединяться непосредственно. Событие в сети Петри – это срабатывание перехода, при котором метка переходит из входной позиции в выходные позиции. Особо следует выделить подкласс WF-сетей [4], который по своей структуре близок к структуре блок-схемы алгоритма.

К достоинствам данной модели следует отнести высокую наглядность функционирования, использование различных типов меток, а также возможность анализа различных ситуаций, возникающих в процессе функционирования и наличие изоморфности процессов, происходящих в вычислительных системах и сетях Петри. Однако, смысловое значение слова «переход» в большей степени предполагает обозначение в виде ребра графа, а не вершины, что несколько затрудняет формализацию процесса построения модели, а значит, и сужает возможности для автоматизации этого процесса, что влечёт за собой сокращение применения этого аппарата для тестирования реального ПО.

Основной целью создания всех перечисленных выше моделей является создание тестов «чёрного ящика». Однако, в большинстве случаев, определение того, что является тестируемым «чёрным ящиком» ложится на тестировщика системы. Также к компетенции тестировщика относится определение того, что является состоянием и переходом в модели тестирования системы. Как правило, тестировщик определяет это, исходя из понимания контекста использования системы. Как следствие, модель тестирования и интерпретация результатов применения этой модели в значительной степени зависят от точности понимания терминологии системы тестировщиком. Это приводит к повышению временных затрат на проведение тестирования и к снижению точности построения тестовых наборов, что может выражаться, например, в обнаружении ошибок уже в ходе промышленной эксплуатации системы.

Одним из возможных решений указанных проблем является применение новых подходов, ориентированных на использование различных методов, в частности ассоциативной памяти [5, 6] и теории концептов [7, 8]. При этом к моделям, применяемым в данных методах выдвигается требование изо-

морфности, т.е. существования функций преобразования исходного программного кода, спецификации и технического задания заказчика в данную модель и однозначность трактовки терминов системы в рамках модели.

## 2. Гибридная ресурсная сеть

Одной из возможных моделей, удовлетворяющей указанным выше требованиям является гибридная ресурсная сеть (ГРС). ГРС представляет собой сеть взаимосвязанных слоёв (сеть слоёв), каждый из которых содержит потоковую модель Форда-Фалкерсона либо однородную ресурсную сеть [5, 6]. Ресурсом в данной сети является тест-пакет. Для формального определения тест-пакета Введём в рассмотрение множество $I_0$ символьных идентификаторов переменных. Идентификатор простого типа определяется по следующей формуле:

$$S_0 : I_0 \to T_i, \qquad (1)$$

где $I_0$ - множество идентификаторов простых типов, $T_i$ $(1 \le i \le n)$ – некоторый фиксированный простой тип данных (n- число типов данных в моделируемом языке программирования). Соответствие $S_0$ (см. формулу (1)) является функциональным, поскольку каждому идентификатору может быть сопоставлен всего один тип.

Однородный сложный тип данных (массив) можно представить следующим образом.

$$S_1 : I_0 \to T_i^k, \qquad (2)$$

где $T_i$ - некоторый фиксированный простой тип данных $(1 \le i \le n)$ – некоторый фиксированный простой тип данных ( n - число типов данных в моделируемом языке программирования) k - количество элементов в массиве.

Сложный неоднородный тип данных (запись) можно представить следующим образом:

$$S_2 : I_0 \to (S_0 \cup S_1)^m, \qquad (3)$$

где $S_0$ - множество, определяемое формулой (1), $S_1$ - множество, определяемое формулой (2), m - количество полей в сложном типе. Примером неоднородной структуры данных может служить тип данных структура (struct) языка С.

Таким образом, множество всех переменных $K$ для языка программирования в дальнейшем будем определять следующим образом:

$$K = S_0 \cup S_1 \cup S_2, \qquad (4)$$

где $S_0$ - множество идентификаторов простых типов, определяемое формулой (1), $S_1$ - множество идентификаторов массивов, определяемое (2), $S_3$ - множество идентификаторов неоднородных структур данных, определяемое (3).

Используя введённые множества, определим далее понятие ресурса в гибридной ресурсной сети. Введём в рассмотрение множество дискретных отсчётов времени $T = \{t_0, t_1, ..., t_n\}$. Обозначим $t_0 \in T$ - время начала тестирования, $t_n \in T$ - окончание времени тестирования. Определим множество всех возможных тест-пакетов следующим образом:

$$P = T \times K^n, \qquad (5)$$

где K - множество, определяемое формулой (4). Элемент $p \in P$ будем называть в дальнейшем тест-пакетом.

Основываясь на понятии тест-пакета и известного понятия графа введём в рассмотрение понятие слоя гибридной ресурсной сети, как упорядоченной двойки множеств:

$$(2^P; E), \qquad (6)$$

где $2^P$ - булеан множества тест-пакетов, E - множество операций над тест-пакетами.

Таким образом, слой гибридной ресурсной сети представляет собой граф, вершинами которого являются множества тест-пакетов, а рёбра – операции на этих тест-пакетах. Принимая во внимание, что тест-пакет для программы определён как вектор идентификаторов и значений переменных программы, можно определить множество рёбер графа E, задаваемого формулой (7), как операции программы. При этом аргументами любой операции программы является множество тест-пакетов. Справедливость этого следует из того, что тест-пакет содержит все переменные программы. При этом, те переменные, которые не изменяются операцией, считаются инвариантными к данной операции.

Из дискретной математики известно, что любому множеству можно поставить в соответствие некоторое целое число, являющейся мощностью данного множества. Учитывая, что имеют место соотношения $(p_1, p_2) \in E; p_1, p_2 \in 2^P$, определим, что:

$$\forall p_1, p_2 : |p_1| + |p_2| = a = \text{const}, \qquad (7)$$

где $|p_1|$, $|p_2|$ - мощности множеств тест-пакетов, находящихся в вершинах графа. a - количество тест-пакетов в сети. Число, эквивалентное мощности множества, находящегося в вершине графа, будем называть яркостью вершины графа ресурсной сети. При этом нулевая яркость вершины (вершина – пустое множество) означает отсутствие данных для обработки операцией, представимой исходящим из данной вершины ребром.

Указанную формулу (7), в силу известных математических тождеств, можно переписать в таком виде:

$$|p_1| + |p_2| = |p_1| + |p_2| + b - b.$$

Величину b представленную в правой части равенства будем называть пропускной способностью ребра. При этом знак «плюс» берётся в том случае, если операция в ребре выполняется в обычном порядке выполнения операций в тестируемой программе, а знак «минус» - если функционирование происходит в обратном направлении. В том случае, если вершине сопоставлены обе пропускные способности, процесс перехода не происходит. Это позволяет учесть в модели как направление обработки данных, так и возможность обратной трассировки.

Для представления программного кода на слое используется граф, схожий по структуре с WF-сетью [4]. Для представления пользовательского интерфейса используется однородная ресурсная сеть.

Определим далее понятие коммуникационного слоя (ребра) гибридной ресурсной сети. Пусть заданы два слоя $L_1 = (2_1^p; E_1)$ и $L_2 = (2_2^p; E_2)$. Коммуникационным слоем $(\{p_1; p_2\}; \{(p_1; p_2)\})$ называется слой, состоящий из двух вершин $p_1 \in 2_1^p$ $p_2 \in 2_2^p$, соединённых направленным ребром. Коммуникационный слой используется в ГРС для моделирования операций вызова процедур между собой и из пользовательского интерфейса.

Основным отличием ГРС от известной модели Крипке является то, что в вершине, фактически, находится набор векторов возможных значений всех объектов предметной области, а не логические высказывания об этих объектах. Вместе с тем, ГРС сохраняет важное достоинство этой модели, по сравнению с обычными конечными автоматами – корректное определение понятия состояния предметной области как множества рассматриваемых в ней объектов и значений их атрибутов. Также, ГРС может быть приведена к модели Крипке на основе известного утверждения о трансформации любого отношения в предикат и операцию [9].

Отличием ГРС от известной модели сетей Петри является отказ от использования двудольного графа. Вместе с тем, понятие ресурса в этой сети является эквивалентным понятию метки в сети Петри.

## 3. Тестирование на основе гибридной ресурсной сети

В теории чисел известно, что любое число в некоторой системе счисления с постоянным или переменным основанием представляет собой упорядоченный n-элементный кортеж $(a_1, a_2, a_3, ..., a_n) \in B$. Будем полагать, что элементы этого кортежа принадлежат некоторому множеству $A$. Введём в рассмотрение функцию вида $f: B \to N$, где $N$ – натуральное число. Аналитически данную функцию возможно представить в следующем виде:

$$f(a_1, a_2, a_3, ..., a_n) = a_1c_1 + a_2c_2 + ... + a_nc_n, \qquad (8)$$

где $a_1, a_2, a_3, ..., a_n$ – соответствующие элементы кортежа из множества $B`$, а $c_1, c_2, c_3, ..., c_n$ – произвольно выбранные натуральные коэффициенты. Результат данной функции будем называть десятичным эквивалентом кортежа $b \in B$, а саму функцию (8) – функцией десятичного эквивалента. Отметим, что десятичный эквивалент не всегда может совпадать со значением числа, определяемого вектором, в силу произвольности выбора коэффициентов *c* в функции. Также это несоответствие между значением вектора и его десятичным эквивалентом подтверждает тот факт, что все существующие на сегодняшний день форматы чисел с плавающей запятой также представляются с использованием целых чисел, а, следовательно, могут быть представлены в виде кортежа $b \in B$, а десятичный эквивалент в этом случае не равен значению, представляемому форматом.

Десятичный эквивалент также может быть вычислен и для символьной системы счисления, где роль цифр играют некоторые символы, однако в этом случае необходимо перед вычислением функции (8) вычислить преобразующую функцию $g: C \to A$, где $C$ - некоторое множество символов. Эта преобразующая функция может задаваться как аналитически, так и таблично. Примерами таких преобразующих функций могут являться таблица ASCII и представление римских цифр в десятичной системе счисления.

Все существующие на сегодняшний день форматы хранения данных основаны на использовании монотонно возрастающих функций. Приведенная функция (1) также является монотонно возрастающей относительно любой переменной, входящей в её запись. Таким образом, при изменении значения переменной изменяется и её десятичный эквивалент.

Очевидно, что в процессе работы программы значения тех или иных полей формата изменяются, а, следовательно, изменяется значение десятичного эквивалента. Обозначим значение функции (8) через d и введём в рассмотрение функцию $\lambda: T \to N$, возвращающую значение десятичного эквивалента в некоторый момент времени. Совпадение значений производных этой функции для программы и спецификации в одной и той же точке означает совпадение поведения программы и спецификации на этих данных, а значит и поведения программы в целом.

При применении указанного метода возникает проблема формирования гибридной ресурсной сети для спецификации системы. В работе [10] приведен один из возможных алгоритмов построения однородной гибридной ресурсной сети для текста на естественном языке. При построении гибридной ресурсной сети на основе спецификации системы производится построение однородной ресурсной сети на основе анализа введённого текста и другой информации. Далее, на основе результата работы этого алгоритма строится представление спецификации в виде слоя ГРС.

Такой подход позволяет в значительной степени решить задачу понимания спецификации предметной области на основе понятий «концепта» и «лексема», приведенных в [8].

## 4. Автоматизированная система тестирования

На основе разработанной модели синтезирована система автоматизированного тестирования «C4ITS» (Components for Improve Testing Software). Это комплексная система тестирования ПО представляет собой набор инструментов и модулей для тестирования ПО, таких как: редактор схемы алгоритмов, редактор UML диаграмм, анализатор исходных кодов программы, модули тестирования программного обеспечения, визуальный редактор графа, редактор организационной структуры предприятия. Одними из ключевых элементов системы являются менеджер задач, управляющий процессом тестирования, редактор графа, позволяющий отследить процессы в менеджере задач и редактор спецификаций на естественном языке, позволяющий заказчику сформулировать требования в понятных ему терминах

## 5. Визуальный редактор графа

Модуль «Визуальный редактор графов» позволяет визуализировать графы, загруженные из XML-документа, редактировать вершины и ребра графа, просматривать и редактировать тестовые пакеты, сохранять граф в документ формата XML.

Для позиционирования вершин при визуализации графа используется силовой алгоритм Фрюхтермана и Рейнгольда [11] с модификацией в виде добавления силы гравитации.

Для оценки качества визуализации используются такие эстетические критерии: Размещение должно было быть как можно более симметричным и равномерным, при этом все вершины, связанные ребром, должны были располагаться на примерно одинаковом расстоянии, называемом «естественной» длиной ребра. Естественная длина ребра рассчитывается по следующей формуле:

$$l = \sqrt{\frac{a}{n}}, \qquad (9)$$

где l – естественная длина ребра,

a — площадь полотна;

n – количество вершин графа.

Идея алгоритма состоит в том, чтобы представить граф (рис. 1) в виде механической системы, где вершины – материальные точки, отталкивающиеся друг от друга, а ребра – пружины, притягивающие друг к другу вершины, которые они связывают.

Таким образом, между каждой парой вершин U и V действуют силы отталкивания:

$$f_{rep}(p_u, p_v) = \frac{l^2}{\| p_u - p_v \|} \overrightarrow{p_u p_v}, \qquad (10)$$

где $f_{rep}$ – сила отталкивания;

$p_u$ – центр вершины U;

$p_v$ – центр вершины V.

А между каждой парой вершин *U* и *V*, связанных ребром – силы притяжения:

$$f_{attr}(p_u, p_v) = \frac{\| p_u - p_v \|^2}{l} \overrightarrow{p_v p_u}, \qquad (11)$$

где $f_{attr}$ – сила притяжения;

$p_u$ – центр вершины U;

$p_v$ – центр вершины V.

Кроме этого, на каждую вершину V действует сила притяжения к центру масс B:

$$F_{grav}(v) = c_{grav}(\overrightarrow{B - p_v}), \qquad (12)$$

где $F_{grav}$ – сила гравитации;

$p_v$ – центр вершины V;

B – центр масс;

$c_{grav}$ – гравитационный коэффициент.

Сила гравитации была введена для того чтобы избежать убегания слабосвязанных вершин к краям изображения.

Чтобы получить конфигурацию равновесия для описанной системы, вершины итеративно сдвигаются в момент времени t в соответствии с вектором равнодействующих сил $F_v(t)$, действующих на вершину. После вычисления $F_v(t)$ для всех вершин каждая вершина сдвигается по этому вектору на величину $\delta(t)F_v(t)$. В этом алгоритме также появляется понятие «температуры» (функция $\delta(t)$), которое используется следующим образом. Температура начинается с начального значения и убывает до нуля. Температура управляет перемещениями вершин так, что по мере того, как улучшается размещение, изменения позиций вершин уменьшаются.

Итеративно вычисляя силы, действующие на каждую вершину и изменяя соответственно позиции этих вершин, система переводится в стабильное состояние, в котором локальные улучшения невозможны (рис. 2).

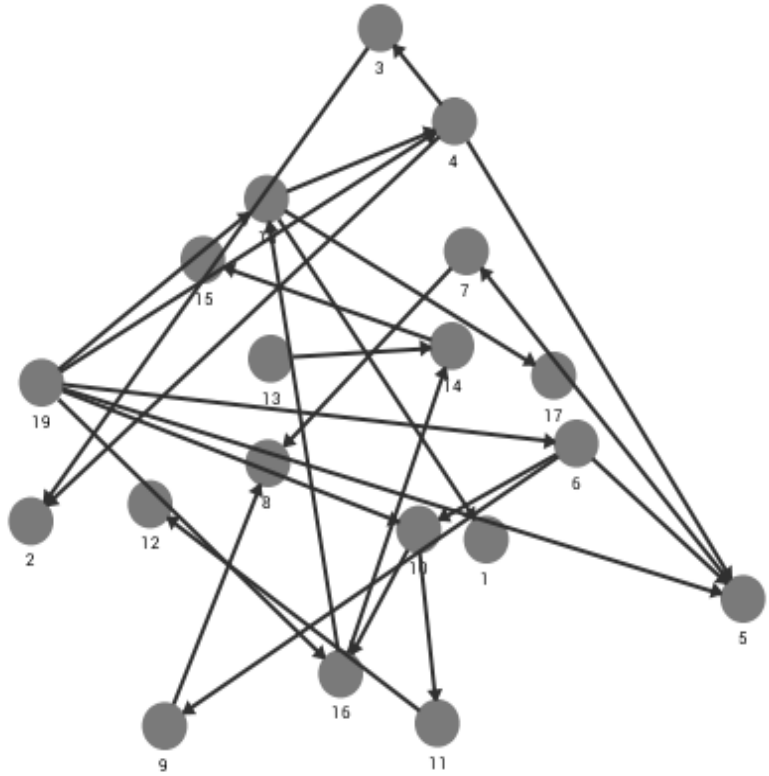


Рис. 1. Граф до автоматического позиционирования

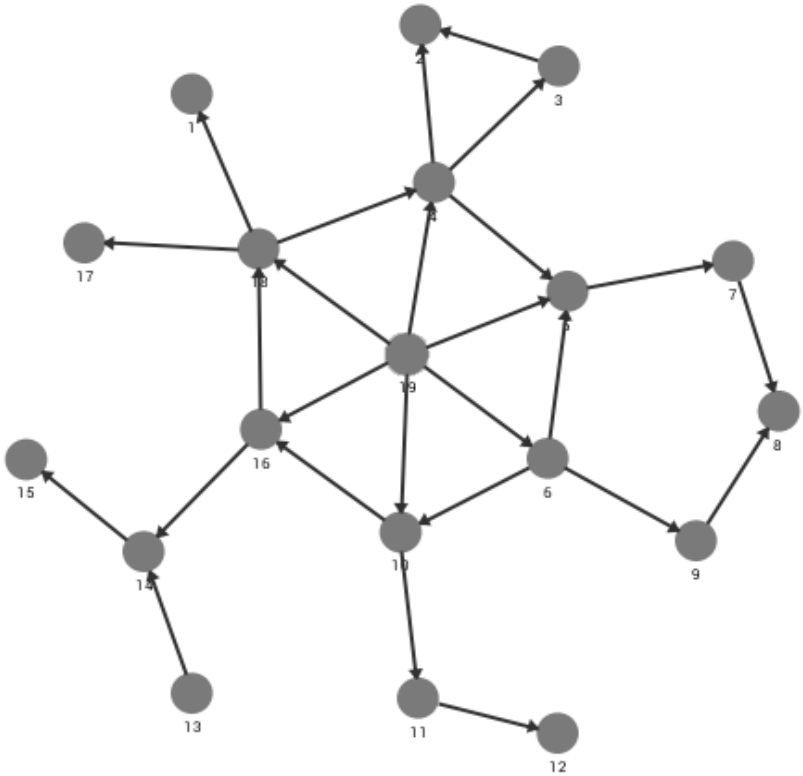


Рис. 2. Граф после автоматического позиционирования

## 6. Архитектура визуального редактора

Для проектирования архитектуры приложения разумно будет воспользоваться шаблонами проектирования. Так, для приложения с графическим интерфейсом пользователя идеально подходит шаблон MVC (Model View Controller – Модель Вид Контроллер).

Для реализации изменений модели можно применить шаблон «Команда», что позволит инкапсу-

лировать каждое изменение модели в отдельном объекте и реализовать механизм отмены и повтора изменений (undo/redo). «Команда» представляет собой объект, который инкапсулирует данные об изменении и исходном состоянии модели и реализует методы выполнения и отмены изменения. Помещая каждую выполненную команду в стек «undo» можно затем отменить все изменения в порядке, обратном выполнению, доставая их по очереди с вершины стека. Если при этом каждую отмененную команду помещать в другой стек «redo», то все отмененные изменения можно вернуть таким же образом.

Итак, в нашем приложении моделью будет выступать граф (Graph), видом – холст (Canvas), на котором будет отображаться граф, контроллером – редактор (GraphEditor), который будет применять к графу различные команды в ответ на события, возникающие при соответствующих действиях пользователя. Различные команды, такие как добавление вершины, добавление ребра, удаление вершины, удаление ребра, перемещение вершины, выполнение симуляции и другие, будут наследовать общий интерфейс Command.

## 7. Менеджер задач

Центральной частью автоматизированной системы тестирования является менеджер задач. Именно он управляет различными частями системы и координирует их работу. В частности, менеджер задач получает задачи на симуляцию исполнения программ и подпрограмм на графах и распределяет эти задачи между клиентскими вычислительными машинами.

Здесь встает проблема распределения задач между клиентами. Следует учесть, как то, что задачи могут быть различной сложности, так и то, что клиентские машины обладают различной производительностью.

Известны два подхода к проблеме диспетчирования. При первом каждая задача из входного набора предварительно назначается для обслуживания на некоторую машину. Задачи начинают выполняться только после окончания диспетчирования всего входного набора. Метод, основанный на таком подходе, называют статическим диспетчированием. Задача статического диспетчирования допускает оптимальное решение, которое в принципе можно получить полным перебором возможных вариантов. Однако с ростом числа вариантов эффект от использования оптимального распределения (по сравнению с некоторым случайным распределением) может быть сведен на нет трудоемкостью получения оптимального результата.

При втором подходе задачи назначаются для выполнения по мере освобождения машин (процессоров) от решения ранее распределенных задач. Здесь проблема, решаемая диспетчером, значительно упрощается, так как будет анализироваться не весь входной набор, а только некоторые подмножества его задач, готовых к выполнению и ожидающих своего распределения. В этом случае будем говорить о динамическом диспетчировании. Применение динамического диспетчирования приводит, как правило, к субоптимальному решению задачи о назначениях. Такое решение получаем, применяя алгоритмы со сравнительно малой трудоемкостью, при этом качество полученного решения незначительно отличается от оптимального.

Для упрощения системы предлагается реализовать менеджер задач в виде веб-сервиса с использованием архитектуры REST [12].

REST (Representational State Transfer) — архитектурный стиль взаимодействия компонентов распределённого приложения в сети. REST представляет собой согласованный набор ограничений, учитываемых при проектировании распределённой системы. Ограничения включают использование клиент-серверной архитектуры, отсутствие состояния (сервер не сохраняет информацию о клиентах между запросами), единообразие интерфейса (каждый ресурс идентифицируется уникальным URI) и некоторые другие. Для взаимодействия используется протокол HTTP.

Данное решение позволит менеджеру задач отказаться от сохранения информации о клиентах и поддержки постоянного соединения с ними, и выдавать задачи только по требованию клиента.

## 8. Редактор спецификаций заказчика

В рамках программного комплекса разработан редактор спецификаций заказчика, который реализует процесс преобразования документации и организационной структуры предприятия – заказчика программного обеспечения в представление на основе гибридной ресурсной сети. Базовый функционал данной части программного комплекса составляют: добавление новых отделов, редактирование существующих отделов, просмотр организационной структуры предприятия, добавление связей между отделами добавление документов на связи между отделами, преобразование полученной архитектуры в потоковый граф.

Схема полученной организационной структуры предприятия, будет хранится в файле, формата xml. Данный xml-файл хранит в себе названия всех отделов, связи между отделами, а там же позиции на экране, для визуализации схемы предприятия. Тег state - содержит в себе название отдела; тег transition

- указывает между какими блоками(отделами) построена связь; тег presentation - содержит в себе координаты блоков (указывающих отдел); тег automaton - содержит в себе информацию об отделах и связях между ними, включая электронные копии документов между отделами. Применение данной подсистемы в рамках системы автоматизированного тестирования позволяет синтезировать ГРС на основе информации о структуре предприятия-заказчика.

## Заключение

Таким образом, в статье решена важная задача повышения точности проведения тестирования за счёт применения автоматизированной системы тестирования на основе новой модели тестирования. Количество сбоев в программной системе, тестирование которой проводилось на основе разработанной системы уменьшилось в среднем на 20%.

В работе впервые предложена модель программного обеспечения на основе ГРС, которая отличается от существующих наличием функционального соответствия понятий «операция» и «функция» в модели и в программной системе.

В работе впервые предложен метод анализа результатов тестирования, основанный на применении понятий ГРС, десятичного эквивалента структуры данных и дифференциального исчисления.

В работе синтезирована архитектура для комплексной системы автоматизированного тестирования ПО «C4ITS», которая базируется на использовании модели ПО, на основе гибридных ресурсных сетей. Рассмотрены одни из основных частей системы: менеджер задач и визуальный редактор графов.

Для визуального редактора предложен алгоритм оптимального позиционирования вершин графа, основанный на силовых алгоритмах Фрюхтермана и Рейнгольда. Предложена реализация менеджера задач в виде RESTful веб-сервиса. Разработан редактор спецификаций со стороны заказчика, позволяющий построить ГРС на основе текстового описания системы

## Литература

*1. Cobb, C. Making Sense of Agile Project Management: Balancing Control and Agility [Text] / C. Cobb. - New Jersey : Wiley, 2011. – 245 p.*

*2. Bloch, M. Delivering large-scale IT projects on time, on budget, and on value. [Electronic resource] / M. Bloch, S. Blumberg, J. Laartz. – Available at: http://www.mckinsey.com/business-functions/business-technology/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value.– 1.04.2016.*

*3. Irlbeck, M. Dependable Software Systems Engeneering [Text] / M. Irlbeck, D. Peled, A. Pretschner. – Washington, Tokyo, IOS Press, 2014. – 320 p.*

*4. Wil, M. P. van der Aalst. Challenges in Business Process Management: Verification of Business Processing Using Petri Nets [Text] / Wil M. P. van der Aalst // Bulletin of the EATCS. – 2003. – Vol. 80. – P. 174-199.*

*5 Kuznetsov, O. P. Flows and Limit States in Bidirectional Resource Networks [Text] / O. P. Kuznetsov, L. Yu. Zhilyakova // Preprints of the 18th IFAC World Congress. Milano (Italy), August 28 - September 2, 2011. – Milano, 2011. - P. 14031-14035.*

*6. Kuznetsov, O. P. Nonsymmetric resource networks. The study of limit states [Text] / O. P. Kuznetsov, L.Yu. Zhilyakova // Management and Production Engineering Review. – September 2011. – Vol. 2, № 3. – P. 33–39.*

*7. Валькман, Ю. Р. Когнитивность семиотики [Текст] / Ю. Р. Валькман // Международная научная конференция ИАИ-2013 : Сборник трудов, Киев 15-17 мая 2013 г. – К., 2013. – С. 69 – 81.*

*8. Валькман, Ю. Р. Концепты: определение, структура, классификация [Текст] / Ю. Р. Валькман // Международная научная конференция ИАИ-2013 : Сборник трудов, Киев 15-17 мая 2013 г. – К., 2013. –С. 82 – 94.*

*9. Кузнецов, О. П. Дискретная математика для инженера [Текст] / О. П. Кузнецов. – М. : Лань, 2009. – 400 с.*

*10. Жилякова, Л. Ю. Поиск в ассоциативной модели памяти [Текст] / Л. Ю. Жилякова // Международная. Научная конференция ИАИ-2009. – Киев, 2009. – С. 124 – 130.*

*11. Fruchterman, T. Graph drawing by force-directed placement [Text] / T. Fruchterman, E. Reingold // Software: Practice and Experience. – 1991. – № 11. – P. 1129-1194.*

*12. Richardson, L. RESTful Web Services [Text] / L. Richardson, S. Ruby. – O'Reilly Media, 2007. – 454 p.*

## References

1. Cobb, C *Making Sense of Agile Project Management: Balancing Control and Agility*. New Jersey, Wiley Publ., 2011. 245 p.

2. Bloch, M., Blumberg, S., Laartz J. *Delivering large-scale IT projects on time, on budget, and on value*. Available at http://www.mckinsey.com/business-functions/business-technology/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value (accessed 1.04.2016).

3. Irlbeck, M., Peled, D., Pretschner, A. *Dependable Software Systems Engeneering*. Washington, Tokyo, IOS Press Publ., 2014. 320 p.

4. Wil M. P. van der Aalst. Challenges in Business Process Management: Verification of Business Processing Using Petri Nets. *Bulletin of the EATCS*, 2003, vol. 80, pp. 174-199.

5. Kuznetsov, O. P., Zhilyakova, L. Yu. Flows and Limit States in Bidirectional Resource Networks. *Preprints of the 18th IFAC World Congress. Milano (Italy) August 28 - September 2, 2011,* pp. 14031-14035.

6. Kuznetsov, O. P., Zhilyakova, L. Yu. Nonsymmetric resource networks. The study of limit states. *Management and Production Engineering Review,* vol. 2, no. 3, September 2011, pp. 33-39.

7. Val'kman, Yu. R. Kognitivnost' semiotiki [cognition semiotics]. *Mezhdunarodnaya. Nauchnaya konferentsiya IAI-2013* [Proc Int conf Intelligent Information Analisys 2013]. Kiev, Prosvita Publ., 2013, pp. 69 – 81 (In Russian).

8. Val'kman, Yu. R. Kontsepty: opredelenie, struktura, klassifikatsiya [Concepts: Determination, Structure Classification] *Mezhdunarodnaya. Nauchnaya konferentsiya IAI-2013* [Proc Int conf Intelligent Information Analisys 2013]. Kiev, Prosvita Publ., 2013, pp. 82 – 94.

9. Kuznetsov, O. P. *Diskretnaya matematika dlya inzhenera* [Discrete mathematics for engeneer]. Moscow, Lan' Publ., 2009. 400 p.

10. Zhilyakova, L. Yu. Poisk v assotsiativnoi modeli pamyati [Search in associative memory model] *Mezhdunarodnaya. Nauchnaya konferentsiya IAI-2013* [Proc Int conf Intelligent Information Analisys 2009]. Kiev, Prosvita Publ., 2009, pp. 124-130.

11. Fruchterman, T. Reingold, E. Graph drawing by force-directed placement. *Software: Practice and Experience*, 1991, no. 11, pp. 1129-1194.

12. Richardson, L. Ruby, S. *RESTful Web Services,* O'Reilly Media Publ., 2007. 454 p.

*Поступила в редакцию 1.04.2016, рассмотрена на редколлегии 14.04.2016*

## АРХІТЕКТУРА СЕРЕДОВИЩА ТЕСТУВАННЯ НА ОСНОВІ МОДЕЛІ ГІБРИДНИХ РЕСУРСНИХ МЕРЕЖ

***О. С. Пригожев, Д. О. Неізвесний, О. С. Ларіонова***

У статті запропонована модель для тестування програмного забезпечення на основі гібридних ресурсних мереж. Визначено поняття десяткового еквіваленту структури даних. Представлений метод проведення тестування на основі гібридних ресурсних мереж, десяткового еквіваленту і диференціального числення. Представлена архітектура системи для автоматизованого тестування і верифікації програмного забезпечення(ПО). Розглянуті одні з основних компонентів системи : менеджер завдань, візуальний редактор графів із застосуванням силових алгоритмів позиціонування вершин графа у візуальному редакторові і редактор специфікації. Спроектована архітектура візуального редактора графів із застосуванням шаблонів проектування. Запропонована реалізація менеджера завдань у вигляді RESTful веб-сервісу.

**Ключові слова:** тестування, ресурсна мережа, автоматизація, розподіл завдань, граф, візуалізація

## ARCHITECTURE MODEL BASED TESTING ENVIRONMENT USE HYBRID RESOURCE NETWORKS

***O. S. Prygozhev, D. O. Neizvesniy, O. S. Larionovy***

In the article offers a software testing model on the basis of hybrid resource networks. The concept of data structure decimal equivalent is certain. The method of model testing software is presented on the basis of hybrid resource networks, decimal equivalent and differential calculation. Architecture of the system the automated testing and verification of software is presented. One of basic components of the system are considered: manager of tasks, visual graph editor with the use of power algorithms for graph vertex positioning in a visual editor and editor of specification. Architecture of visual graph editor is projected with the use of templates of planning. offers realization task manager as RESTful of web service

**Keywords**: model based testing, resource network, automation, visualization, task manager, hybrid resource network.

**Пригожев Александр Сергеевич** – канд. техн. наук, доцент кафедры системного программного обеспечения, Одесский национальный политехнический университет, Одесса, Украина, e-mail: Oleksandr.prigozhev@gmail.com.

**Неизвестный Дмитрий Александрович** – студент кафедры системного программного обеспечения, Одесский национальный политехнический университет, Одесса, Украина, e-mail: d.unknown07@gmail.com.

**Ларионова Оксана Сергеевна** – студентка кафедры системного программного обеспечения, Одесский национальный политехнический университет, Одесса, Украина, e-mail: oksana_larionovy@mail.ru.

**Prygozhev Oleksandr Sergeevych** – PhD, assistant professor system software chair, Odessa national polytechnic university, Odessa, Ukraine, e-mail: Oleksandr.prigozhev@gmail.com.

**Neizviesny Dmytro Oleksandrovych** – four year student system software chair, Odessa national polytechnic university, Odessa, Ukraine, e-mail: d.unknown07@gmail.com.

**Larionova Oksana Sergeevna -** four year student system software chair, Odessa national polytechnic university, Odessa, Ukraine, e-mail: oksana_larionovy@mail.ru.