УДК 004.94

РАЗРАБОТКА АЛГОРИТМА САМООРГАНИЗАЦИИ ИНТЕРАКТИВНЫХ КОМПОНЕНТОВ ВИРТУАЛЬНОЙ ОБРАЗОВАТЕЛЬНОЙ СРЕДЫ

Волчихин В.И., Бершадский А.М., Бождай А.С., Евсеева Ю.И., Гудков А.А.

ФГБОУ ВПО «Пензенский государственный университет», Пенза, e-mail: president@pnzgu.ru

Современная виртуальная образовательная среда активно использует различные принципы самоорганизации собственных компонентов. Изменчивости в первую очередь подвержены интерактивные компоненты образовательной среды – различные тестирующие приложения, виртуальные тренажеры и т.д. В связи с этим обретает актуальность вопрос разработки эффективной адаптационной стратегии самоорганизации подобных компонентов. Для решения поставленной задачи был применен математический аппарат теории графов и гиперграфов, методы морфологического анализа и синтеза сложных систем, методы моделирования изменчивости программных систем. В результате был разработан алгоритм автоматического ситуационного выбора конфигураций интерактивного компонента виртуальной образовательной среды, обладающего свойством адаптивности. Разработанный в ходе работы алгоритм, в отличие от известных, использует трехконтурную систему обратных связей, регулирующих текущее состояние программы в зависимости от действий пользователя, состояния среды выполнения и доступных аппаратных ресурсов.

Ключевые слова: адаптивное обучающее программное обеспечение, теория графов, ориентированный гиперграф, автоматизированное проектирование, моделирование изменчивости, виртуальная образовательная среда

DEVELOPMENT OF SELF-ORGINISING ALGORITHM OF INTERACTIVE SOFTWARE COMPONENTS OF VIRTUAL LEARNING ENVIRONMENT

Volchikhin V.I., Bershadskiy A.M., Bozhday A.S., Evseeva Yu.I., Gudkov A.A.

Federal state budgetary educational institution of Higher Education Penza State University, Penza, e-mail: president@pnzgu.ru

Modern virtual educational environment uses various self-organization principles of its own components. Variability is using by interactive components of the educational environment, such as various test applications, virtual simulators, etc. In this regard, the question of effective adaptation strategies development of these components self-organization is very urgent. To solve this problem the mathematical apparatus of graphs and hypergraphs theory, methods of morphological analysis and synthesis of complex systems and methods of variability modeling of software systems were applied. Algorithm of automatic situational selection of interactive software component configuration in virtual learning environment was developed. Developed algorithm, in contrast to the well-known, uses three-loop feedback system, that regulate the current state of the program, depending on the user's actions, environment state and the status of implementation of available hardware resources.

Keywords: adaptive educational software, graph theory, oriented hypergraf, computer-aided design, variability modeling, virtual learning environment

В современной виртуальной образовательной среде большую роль могут играть компоненты, обладающие свойствами адаптивности и интерактивности. К таким компонентам относятся различные обучающие и тестирующие программы, виртуальные тренажеры. По этой причине актуальным является вопрос разработки новых адаптационных стратегий самоорганизации таких компонентов. К числу наиболее существенных вопросов, требующих решения, относится вопрос создания алгоритма, реализующего адаптацию программы на основе многоконтурных обратных связей. Помимо регуляции состояния программного компонента в зависимости от действий пользователя, в ряде случаев может быть необходим учет состояния среды выполнения и доступных аппаратных ресурсов. В данной работе приведено описание алгоритма, учитывающего все обозначенные факторы.

Структура, математическая модель и способ создания адаптивного программного компонента виртуальной образовательной среды

Создание алгоритма, эффективно реализующего адаптацию по нескольким контурам обратной связи, требует особого способа организации адаптивной программы. Первоочередная задача – разработка математической модели программы, которая позволит описать основные компоненты программы, взаимосвязи между ними, а также процессы, протекающие в программной системе, в структурированной форме, удобной для последующей алгоритмизации.

С целью создания такой модели было решено применить методы морфологического анализа и синтеза [1]. Результатом выполнения этапа морфологического анализа является морфологическое множество, называемое также множеством альтернатив,

содержащим в себе описание всех возможных вариантов системы. В процессе морфологического синтеза осуществляется поиск на морфологическом множестве оптимальной структуры синтезируемого объекта.

Для задания морфологического множества, описывающего структуру адаптивной программы, используются диаграммы характеристик – расширенные И/ИЛИ-деревья [2, 3]. Пример структуры адаптивной программы в формате диаграммы характеристик представлен на рисунке.

Для обеспечения удобства последующей алгоритмизации и обработки было принято решение использовать гиперграфовое представление описания структуры, заданного в форме диаграммы характеристик. При данном преобразовании к основным элементам диаграммы характеристик будут применяться следующие правила [4]:

1. Множество характеристик модели будет отображено на множество вершин соответствующего гиперграфа *Features* → *V*.

2. Множество взаимоотношений модели будет отображено на множество гиперребер, соединяющих вершины гиперграфа (характеристики модели) *Relations* → *E*.

В данной работе предлагается следующая математическая модель:

$$M = (F, S, X),$$

где F – представление структуры адаптивной программы в форме ориентированного гиперграфа, полученного на основе диаграммы характеристик;

$S = \{S_1, S_2, ..., S_k\}$ – конечное множество конфигураций диаграммы характеристик, каждая из которых является описанием определенного состояния адаптивной программы (подграфом исходного гиперграфа);

X – матрица переходов между состояниями программы.

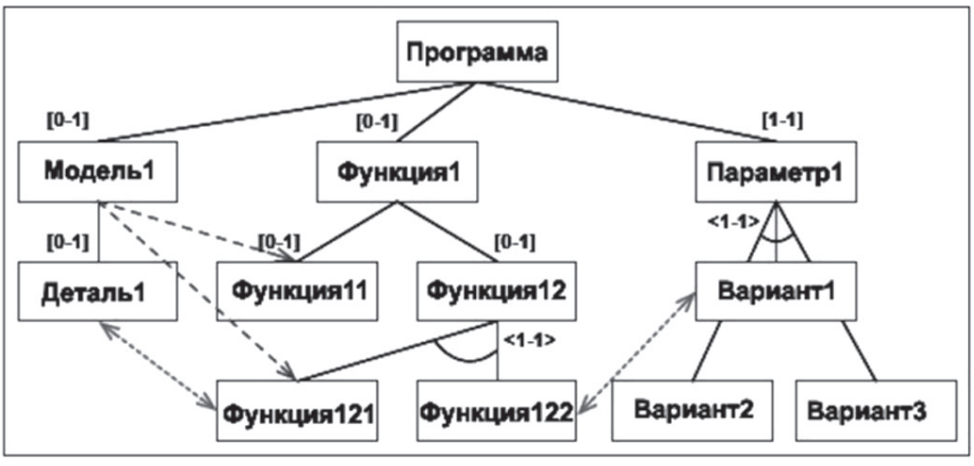
Как видно, предложенная математическая модель изменчивости является достаточно наглядным и удобным инструментом синтеза структуры адаптивной программной системы. Элементы данной модели служат для описания пространства возможных состояний программы.

Метод синтеза адаптивного программного компонента, базирующийся на предложенной математической модели, включает в себя следующие этапы:

1. Этап морфологического анализа адаптивного программного компонента. В ходе выполнения этапа разрабатывается структура программы в формате диаграммы характеристик, выполняется определение пространства параметров элементов диаграммы.

2. Этап теоретико-множественного преобразования структуры программного компонента. Результатом выполнения этапа является гиперграфовое представление полученной на предыдущем этапе структуры.

3. Этап генерации состояний адаптивной программной системы. Результатом выполнения этапа является множество конфигураций исходной диаграммы характеристик, описывающей структуру программы. В процессе разработки программной системы пользователем вручную определяются необходимые структурные конфигурации, являющиеся базовыми для синтезируемой структуры программы. В процессе выполнения на основе имеющихся базовых конфигураций формируются производные промежуточные конфигурации. Данный процесс является частью более широкого процесса адаптации в ходе выполнения программной системы.



Структура адаптивной программы в формате диаграммы характеристик

4. Этап верификации системных пользовательских конфигураций. Основное назначение этапа – поиск и исправление ошибок в системных конфигурациях, описывающих отдельные состояния адаптивной программной системы.

5. Этап определения взаимосвязей между состояниями. Результатом выполнения данного этапа является матрица переходов.

Предложенный метод синтеза адаптивных программных компонентов виртуальной образовательной среды, основанный на гиперграфовом подходе к формализации диаграмм характеристик и построению на их основе системных конфигураций, позволяет формализовать сложную математическую процедуру задания изменчивости наглядным, простым и интуитивно понятным образом с использованием средств визуального проектирования.

Алгоритм автоматического ситуационного выбора системных конфигураций в процессе выполнения

В общем случае об адаптивности в программной системе можно говорить тогда, когда использование текущей информации в ней приводит к изменению алгоритма функционирования. Адаптивную систему, если структура алгоритма в ней не изменяется, а изменяются только параметры, называют самонастраивающейся. Если изменяется структура системы, то ее называют самоорганизующейся. В такой системе процессы самоорганизации присутствуют на всех последующих этапах жизненного цикла [5, 6, 7].

Специфика функционирования адаптивного программного компонента подразумевает наличие трех контуров обратной связи в программной системе:

1) контур, реализующий адаптацию в зависимости от действий пользователя;

2) контур, реализующий адаптацию в зависимости от состояния среды выполнения;

3) контур, реализующий адаптацию в зависимости от доступных аппаратных ресурсов.

Адаптация в соответствии с доступными аппаратными ресурсами является, по сути, параметрической адаптацией. Данный тип адаптации особенно актуален для программного обеспечения, использующего трехмерную графику (например, виртуальных тренажеров) и подразумевает влияние производительности системы на степень детализации геометрических объектов. Степень геометрической детализации, в свою очередь, является параметром соответствующего элемента модели характеристик. Таким образом по данному контуру управления осуществляется процесс самонастройки программы.

Рассмотрим реализацию необходимого алгоритма применительно к программным компонентам, использующим трехмерную графику. Предлагаемый алгоритм ситуационного выбора конфигураций в данном случае включает в себя следующие шаги:

1. Установка номера текущей конфигурации $i = 1$.

2. Выбор конфигурации S_i.

3. Вычисляется показатель P производительности вычислительной системы

$$P = \begin{cases} \dfrac{P_R}{P_E}, \text{ если } P_R \le P_E, \\ 1, \text{ если } P_R > P_E \end{cases},$$

где P_R – производительность ЭВМ, на которой запускается программа;

$P_R = k \cdot P_0$ – производительность эталонной ЭВМ;

k – коэффициент, определяемый автором программы (по умолчанию $k = 1$);

P_0 – производительность ЭВМ, на которой создавался программный компонент.

4. Первый этап параметрического синтеза программного компонента: на основе полученного коэффициента производительности системы происходит выбор соответствующего уровня детализации трехмерных моделей. На диаграмме характеристик узел, соответствующий трехмерному объекту, может содержать несколько геометрических моделей разной степени детализации. Они упорядочены по возрастанию уровня сложности. Если некоторый объект O имеет n уровней детализации $(O_1, O_2, ..., O_n)$, то при визуализации данного объекта будет выбран вариант O_i, где $i = \lceil P \cdot n \rceil$, $\lceil x \rceil$ – целое число, ближайшее к x, $\lceil x \rceil \ge x$.

5. Развертывание поддеревьев моделей, функций и параметров. Ожидание завершения состояния.

6. Осуществляется вычисление показателя качества работы пользователя. При $d > 0$ формула расчета будет иметь следующий вид:

$$C = \frac{\left(\dfrac{t_{norm}}{t} + \dfrac{d_{norm}}{d} \right)}{2},$$

где t – время пребывания системы в текущем состоянии;

d – количество ошибок, случившихся при выполнении шага;

t_{norm} и d_{norm} – нормальные значения показателей t и d, задаваемые автором приложения

для каждой конфигурации. В случае если $d = 0$, формула заменяется на следующую:

$$C = \begin{cases} \dfrac{\left(\dfrac{t_{norm}}{t} + 1\right)}{2}, \text{если } d_{norm} = 0, \\ \infty, \text{если } d_{norm} > 0. \end{cases}$$

7. Второй этап параметрического синтеза программного компонента: определение значений параметров, используемых в функциях. Параметры могут принадлежать к одному из трех типов – целое число, вещественное число, перечисление. В первых двух случаях автор должен задать пару граничных значений ($x_{нач}$, $x_{кон}$), в пределах которых параметр может принимать свои значения. В последнем случае автор должен перечислить значения $x_1, x_2, ..., x_n$. Для расчета параметров необходимо провести вычисление нормализованного показателя $C_{норм} \in [0;1]$. Для начала нужно отобразить интервал $[0;\infty]$, которому принадлежит C, на интервал [0; 1], например, с помощью следующего преобразования:

$$C_{норм} = \begin{cases} \dfrac{C}{b}, \text{если } C \le b, \\ 1, \text{если } C > \text{b}. \end{cases}$$

где $0 < b < \infty$ – наибольшее значение показателя C, определенное для текущего состояния. Другой возможный вариант преобразования выглядит следующим образом:

$$C_{норм} = 1 - e^{kC},$$

где $k > 0$ – некоторый коэффициент, позволяющий настроить вид преобразования. После того как получено нормированное значение показателя работы пользователя с состоянием, вычисляются значения параметров, используемых в следующем состоянии. В случае если параметр является целым или вещественным числом, для которого заданы граничные значения ($x_{нач}$, $x_{кон}$), то значение параметра находится по формуле

$$x = x_{нач} + C_{норм} \cdot (x_{кон} - x_{нач}).$$

Если параметр является перечислением, то его значение равно $x = x_i$, где $i = \lceil C_{норм} \cdot n \rceil$, $\lceil x \rceil$ – целое число, ближайшее к x, $\lceil x \rceil \ge x$.

8. Если это необходимо, корректировка матрицы переходов с учетом состояния среды выполнения

$$X = E \cdot X_{эт},$$

где $X_{эт}$ – исходная матрица переходов, задаваемая пользователем для $E = 1$;
E – показатель, определяющий состояние среды выполнения. Данный показатель вычисляется на основании множества значений показателя C, характеризующих то, как пользователь работал с несколькими последними состояниями программы. В случае если среда выполнения оказывается слишком сложной для пользователя (значения показателя C по результатам прохождения нескольких последних состояний являются низкими), то происходит уменьшение значения показателя E и, соответственно, сужение диапазонов значений, расположенных в ячейках матрицы. При усложнении среды выполнения происходит увеличение значения показателя E.

9. Если существует такой интервал x_{ij}, что $C \in x_{ij}$, то $i = j$ и осуществляется переход к пункту 2 алгоритма. Иначе работа программы завершается.

Заключение

В работе был предложен алгоритм автоматического ситуационного выбора системных конфигураций в процессе выполнения адаптивного программного компонента виртуальной образовательной среды, который, в отличие от известных, использует трехконтурную систему обратных связей. Данные связи регулируют текущее состояние программы в зависимости от действий пользователя, состояния среды выполнения и доступных аппаратных ресурсов.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 15-07-01553.

Список литературы

1. Кумунжиев К.В. Теория систем и системный анализ: учебное пособие / К.В. Кумунжиев. – Ульяновск: УлГУ, 2003. – 240 с.

2. Финогеев А.А. Анализ информационных рисков в системах обработки данных на основе туманных вычислений / А.А. Финогеев, А.Г. Финогеев, И.С. Нефедова // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2016. – № 2. – С. 5–16.

3. Baresi L. Dynamically evolving the structural variability of dynamic software product lines / L. Baresi, C. Quinton // Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. – New York: ACM, 2015. – P. 57–63.

4. Kang K.C. Feature-oriented domain analysis (FODA): feasibility study / K.C. Kang [et al.]. – Pittsburgh: Software Engineering Institute, 1990. – 161 p.

5. Meng A.C. On evaluating self-adaptive software / A.C. Meng // Self-adaptive software / edited by P. Robertson, H. Shrobe, R. Laddaga. – Heidelberg: Springer Berlin Heidelberg, 2001. – P. 65–74.

6. Shen L. Towards feature-oriented variability reconfiguration in dynamic software product lines / L. Shen [et al.] // Top productivity through software reuse / edited by K. Schmid. – Heidelberg: Springer Berlin Heidelberg, 2011. – P. 52–68.

7. Sinnema M. Classifying variability modeling techniques / M. Sinnema, S. Deelstra // Information and software technology. – 2007. – № 7. – P. 42–54.