УДК 004.4'244
DOI: 10.15827/0236-235X.121.184-191
Дата подачи статьи: 09.08.17
2018. Т. 31. № 1. С. 184–191

КОНЦЕПТУАЛЬНЫЕ ОСНОВЫ ПОСТРОЕНИЯ СБОРОЧНОГО ГЕНЕРАТОРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СЛОЖНЫХ СИСТЕМ

В.О. Георгиев [1, 2], *к.т.н., эксперт РАН, старший преподаватель, VOGeorgiev@kpfu.ru*
Н.А. Прокопьев [2], *ассистент, nikolai.prokopyev@gmail.com*
Д.С. Поликашин [2], *аспирант, nullexception@yandex.ru*

[1] *Российская академия наук, Ленинский просп., 14, г. Москва, 119991, Россия*
[2] *Казанский (Приволжский) федеральный университет, ул. Кремлевская, 18, г. Казань, 420008, Россия*

В статье представлены разработанные концептуальные основы построения учебно-макетных вариантов интерактивной сборочной системы, осуществляющей в автоматическом режиме сборку программных систем прикладного назначения.

Генерация программных систем построена на основе концепций «Теории схем программ», ее базисных понятий, типовых конструкционных схем и методологий программной инженерии. Сборка осуществляется в интерактивном взаимодействии с пользователем на профессиональном языке общения. Этот язык построен на основе понятия интеллектуального интерфейса и реализуется на сочетании жестко запрограммированного и свободного сценарного диалогов общения.

Система предназначена для наглядной практической демонстрации основных этапов и работ при разработке сложных программных средств в соответствии с макетно-модельным подходом к программной инженерии.

В статье также рассматриваются формулировки и решения некоторых важных проблем, возникающих при генерации ПО сложных систем, таких как совместимость программных модулей, формализация компьютерного интерактивного взаимодействия и выбор формальных моделей человеко-машинного интерфейса. Предлагается архитектура системы, позволяющая расширять систему плагинами для обработки при сборке несовместимых программных модулей, написанных на разных языках и требующих различные версии библиотек для приведения их к совместимому виду. Систематизируются модели интерактивного взаимодействия, как формальные, так и неформальные. Приводятся макетные реализации системы для трех выбранных моделей: модели на основе теории игр, модели на основе тензорной сети и графовой модели.

Ключевые слова: *генератор ПО, преобразование кода, учебная модель, интерактивная система, технологии и инструменты.*

В данном исследовании поставлены задачи систематизации результатов, полученных в ходе лекционных и практических занятий, а также задача обобщения и использования наработок по формальным моделям и методам представления знаний предметных областей интерактивных систем.

Представляемые макетные варианты системы предназначены для наглядной практической демонстрации содержания основных этапов и работ при разработке сложных, критических программных средств. К основным этапам отнесены проектирование и управление проектом (административное, автоматическое или автоматизированное), тестирование создаваемой системы, ее документирование, оценка качества, оценка стоимости.

Сборка осуществляется в интерактивном взаимодействии с пользователем на профессиональном языке общения. Этот язык построен на основе понятия интеллектуального интерфейса и реализуется на сочетании жестко запрограммированного диалога (на верхнем уровне перехода от одного этапа к другому) и свободного сценарного диалога общения (на нижних, внутриэтапных уровнях).

Программная реализация сценарного диалога построена на основе макетного кодового представления основных формальных моделей интерактивных диалоговых систем человеко-машинного взаимодействия [1, 2].

Сборка может осуществляться в режиме исходных кодов, режиме исполняемых кодов и в режиме сочетания исходных и исполняемых кодов.

Генерация программных систем построена на основе использования концепций «Теории схем программ», ее базисных понятий типовых конструкционных схем и методологий программной инженерии. Программная реализация системы сделана в нескольких вариантах в зависимости от того, какая формальная модель сценария диалога заложена. В качестве формальных моделей выделены графовая (автоматная) модель, модели на основе функциональных сетей Петри, на основе семантических сетей, на основе реляционных данных, на основе теории формальных языков, вероятностная модель, модель на основе теории игр, тензорная модель и некоторые другие.

Система реализована на основе использования основных инструментальных средств языковых сред C# и JAVA.

Систематизация концептуальных основ технологии разработки генераторов ПО сложных систем и методологий их создания

При проектировании такого ПО, как сборочный генератор сложных систем, который сам является

сложной программной системой, возникают многочисленные задачи моделирования. На этапах разработки технического и рабочего проектов систем модели отдельных подсистем детализируются и моделирование служит эффективным инструментом для решения конкретных задач проектирования, то есть выбора оптимального варианта решения задачи по определенному критерию при заданных ограничениях. При моделировании сложных систем необходимо учитывать следующие их особенности:

– сложность структуры и запутанность связей между элементами, неоднозначность алгоритмов поведения при различных условиях;

– большое количество параметров и переменных, неполнота и недетерминированность исходной информации;

– разнообразие и вероятностный характер воздействий внешней среды.

Одним из систематических дисциплинированных измеримых подходов к разработке программных систем в программной инженерии можно назвать макетно-модельный подход, основанный на технологических принципах [3, 4].

Макетно-модельный подход. Макетно-модельный подход заключается в использовании методологии механизма теории схем программ с выделением типовых схем функционального взаимодействия элементов ПО с полной или частичной формализацией их функционирования и программной реализацией макетных моделей элементов ПО. При практической программной реализации используется метод, позволяющий следующее.

1. Определение цели задачи и углубление в ее суть.

2. Установление подцелей и разбиение задачи на подзадачи.

3. Выявление специфических особенностей разрабатываемого для решения задачи ПО.

4. Поиск эксперта, специализирующегося в области, к которой относится задача, и получение его согласия на помощь в разработке системы.

5. Работа вместе с экспертом над несколькими экспериментальными примерами прикладных задач, которые должна решать разрабатываемая программная система.

6. Выбор программных средств, необходимых для создания системы. Этот выбор зависит от типа решаемой задачи, финансовых возможностей и сложности ПО.

7. Выбор технологических средств, на которых будет работать программный инструментарий, приобретение инструментальных средств.

8. Построение лабораторного прототипа системы, позволяющего успешно справляться с примерами задач, решенных в пункте 5.

9. Проектирование системы при участии эксперта. Выявление сущностей, взаимосвязей между ними, виды иерархий, классов.

10. Выполнение необходимого числа итераций разработки системы с испытанием в реальных практических условиях.

11. Разработка документации к системе.

С первых шагов реализации проекта необходимо стремиться к построению хотя бы ограниченной по возможностям, но правильно работающей макетной модели для улучшения взаимодействия с экспертом, для чего и используется макетно-модельный подход.

Проблемы совместимости программных модулей

Существуют нерешенные проблемы, возникающие при разработке ПО сложных систем.

Во-первых, отсутствует обратная совместимость между различными версиями программных компонентов и модулей, входящих в систему, что может повлечь за собой следующее:

– нарушение целостности и единства экосистемы, выстроенной вокруг этого программного компонента, и замедление темпов ее развития из-за разобщения сообщества программистов, использующих в своих разработках несовместимые версии данного ПО;

– замедление реализации и развития проектов, использующих данное ПО;

– проблемы безопасности и эффективности работы системы из-за использования в разработке устаревших программных компонентов;

– экономические издержки при переходе программной системы на новую версию программного компонента.

Во-вторых, в работе генератора ПО сложных систем задействованы многие программные компоненты в виде исходных кодов. В их составе могут быть два или более компонентов, имеющих в своих зависимостях один и тот же модуль, реализованный в разных несовместимых между собой версиях.

Таким образом, генератор ПО сложных систем должен включать возможность решения проблемы совместимости программных модулей.

Решение проблем совместимости. По результатам исследования [5] выявлены четыре класса проблем, наиболее часто встречающихся при осуществлении ручной миграции ПО на новую версию библиотеки:

– изменение сигнатуры программных компонентов с сохранением их семантики;

– изменение количества аргументов функций;

– изменение требований к инициализации или предварительной настройке окружения;

– отсутствие доступного описания всех нюансов перехода на новую версию.

Таким образом, в процессе разработки ПО, нацеленного на решение проблемы совместимости, необходимо учитывать источники исходных дан-

ных, синтаксис языков программирования, фреймворки и специализированные программные библиотеки, форматы представления документации, классы проблем, приводящих к потере совместимости между версиями библиотек, алгоритмы поиска и устранения проблем совместимости.

Поэтому специализированные программные средства – приложения должны удовлетворять условиям расширяемости и гибкости.

Немаловажным аспектом является обеспечение эффективности системы, ее надежности и производительности.

Архитектура специализированных средств сборочного генератора. Для удовлетворения требований выбрана микроядерная архитектура приложения [6], состоящая из основной системы (ядра) и плагинов (рис. 1). Ядро содержит минимум логики основного функционала, но руководит загрузкой, выгрузкой и запуском необходимых плагинов, а плагины не связаны друг с другом промежуточными зависимостями.

Производительность приложения, построенного на основе такой архитектуры, напрямую зависит от количества подключенных и активных модулей. При необходимости большей производительности такая архитектура легко модифицируется в микросервисную путем вынесения основной логики из плагина в отдельный сервисный процесс. В таком случае плагин используется только для связи сервиса с ядром программы.

Ядро системы обеспечивает командный интерфейс для настройки и действия по поиску и загрузке плагинов из внешнего хранилища (функции менеджера пакетов). Основным предназначением ядра является работа с загруженными плагинами: установка, включение, отключение и удаление, конфигурирование, координация работы плагинов между собой, взаимодействие плагинов и пользователя.

По выполняемым задачам плагины делятся на следующие виды.

1. Плагины, ориентированные на выявление проблем совместимости на основе подхода, изложенного в [7, 8]. На вход получают различные версии библиотеки, между которыми необходимо осуществить миграцию целевого ПО. В данный вид входят плагины, зависящие от синтаксических конструкций (языка, фреймворка, формата конфигурации) [9], и плагины, обрабатывающие специально подготовленные абстрактные модели.

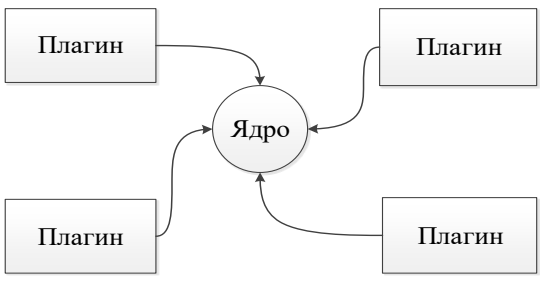

Рис. 1. Схема микроядерной архитектуры

Fig. 1. A microkernel architecture scheme

2. Плагины, ответственные за устранение проблем совместимости в целевом коде. Данные плагины всегда зависят от синтаксиса обрабатываемой структуры [10] и направлены на приведение рекомендаций и указаний для осуществления миграции программы между версиями библиотеки самим программистом и на проведение автоматизированной трансформации исходного кода целевой программы и конфигурационных файлов системы сборки ПО согласно подходу, изложенному в [11].

3. Плагины, ответственные за взаимодействие с пользователями.

Проблемы формализации компьютерного интерактивного взаимодействия

Современный этап развития программных средств характеризуется широким распространением интерактивных (диалоговых) систем, обеспечивающих решение различных классов задач на основе оптимального разделения функций между пользователем и вычислительной машиной. Важным аспектом диалоговой системы является возможность многовариантной, гибкой и вместе с тем простой и удобной для пользователя настройки программы. При построении человеко-машинного интерфейса генератора был выбран формат *диалоговой системы* (ДС).

Модели ДС. Среди основных и обязательных этапов построения ДС существенная роль отводится выбору и разработке формализованной модели системы, позволяющей упорядочить описание ДС, облегчить задачу их проектирования и анализа, служить концептуальной основой анализа их поведения и реализации.

Модель ДС представляет собой описание схем диалоговых процессов и действий (функций), выполняемых системой, и должна удовлетворять следующим условиям:

– быть простой и выразительной;

– обладать возможностью абстрагироваться от содержательной части диалога (от разделения ролей между человеком и ДС);

– быть доступной для формально-математической трактовки;

– предоставлять возможность преобразования в программную систему.

Применяемые в настоящее время методы и средства построения моделей ДС можно разделить на неформальные (или частично формализованные) и формальные.

Неформальные методы для своего определения в основном не используют никаких специализированных формальных нотаций. В качестве методов и средств построения неформальных моде-

лей используются блок-схемы (схемы диалога), диаграммы состояний, табличные формы описания, тексты на языках программирования высокого уровня, тексты на языке некоторой виртуальной машины, методы структурного программирования.

При использовании ***формальных методов построения моделей*** ДС применяются определенные специализированные формальные нотации.

Выделим два направления в таких формализациях: создание моделей для описания организации или функционирования, анализа, оценки и оптимизации ДС, а также подход к исследованию взаимодействия человека и ЭВМ, заключающийся в определении типов решаемых задач, в выделении абстрактных средств их решения и анализе различных видов взаимодействия человека и ЭВМ.

Выделим следующие формальные модели ДС (подробнее см. [12–15]): графовые (автоматные) модели; модели, использующие сетевое представление (сети Петри, сети Вудса и обобщенные сети перехода); модели, построенные на основе аппарата формальных грамматик; модели, использующие аппарат теории задач и способов их решения; вероятностные модели; модели, использующие понятие процесса; модели, основанные на различных методах формальных спецификаций; реляционные и фреймово-реляционные модели; модели, построенные на основе операторных схем; модели, использующие принципы теории игр; модели, основанные на использовании аппарата каркасов теории моделей; естественно-языковые модели.

Практическая кодовая реализация макетов генераторов ПО сложных систем с разными моделями

Макет на основе модели, использующей принципы теории игр. Данная модель [16] не является самой распространенной формальной моделью, однако она достаточно проста в использовании и имеет немало сходств с наиболее распространенной автоматной моделью.

Теория диалоговых игр – это попытка учесть идеи сценарных моделей и диалоговых грамматик в одной структуре. В этой структуре ходы часто приравниваются к речевым актам. Модель довольно формально определяет ходы, допустимые для каждого из участников в данный момент игры (по правилам и в соответствии с целью), и таким образом моделируются диалоги.

Для задания требований к диалоговой системе вводятся следующие определения.

Игровая доска определяется двумя ее частями: общая доска и личные доски.

Диалог принято представлять как взаимодействие на *общей доске*. Эта доска содержит упорядоченную историю ходов, идентификатор игры со связанными правилами, которые были известны до начала или стали известны в процессе диалога.

Личная доска – личные представления каждого игрока, неизвестные другим. Это, в частности, предполагает игру с неполной информацией.

Ход – базовое действие игрока в игре, имеющее цену. Шагом игры называется интервал между двумя ходами. Игрок, который должен ходить, является активным.

Когнитивный контекст – состояние личных досок и совместно принятые факты на публичной доске. С каждом шагом игры меняется ее когнитивный контекст из-за добавления или удаления фактов на игровой доске. Таким образом, это понятие может быть отнесено к акту диалога. Формально акт диалога определен как коммуникативная функция, примененная к содержательным высказываниям. Также вводится пустой ход. Он не влияет на когнитивный контекст игры, но увеличивает счетчик ходов, как любое диалоговое действие.

Вдобавок активный игрок может сделать логический ход. Такой ход изменяет доску при применении логического правила. Логический ход ничего не предлагает и не увеличивает счетчик игры.

Цели. Даже если намерения напрямую не реализованы в игре, игроки преследуют в ней цели. Цели записаны в данной модели структурой игры.

Правила. Понятие «правило» сильно связано с сущностью игры. Нет подхода, который мог бы дать чистое представление различных правил в диалоговых играх. Начальные же правила игры можно просто считать функцией, определяющей доступные ходы. Такие правила называются эффективными.

Структуру комплекса составляют две части: генератор сценариев и диалоговая система. В практической реализации модель выглядит следующим образом.

- Имеется история диалога, в которой хранятся все фразы системы и ответы пользователя.
- Имеется функция, принимающая в качестве аргумента историю диалога и возвращающая множество допустимых ходов. Эта функция определяет правила игр.
- Каждый речевой акт заносится в историю. Если какой-то шаг не удовлетворяет правилам игры, состояние игры откатывается на шаг назад.

Создание сценария (рис. 2), построенного на основе модели теории игр, начинается добавлением вопросов и прочей информации и завершается созданием конфигурационного документа сценария. Эксперт заполняет текст вопроса, правила в таблице, переход на другой сценарий диалога, справку и сообщение об ошибке (см. http://www.swsys.ru/uploaded/image/2018_1/2018-1-dop/27.jpg). Если оставить поле «Переход» пустым, то при соответствующем ему ответе система автоматически перейдет к следующему вопросу (игрок в таком случае совершает пустой ход).

После выполнения этих действий созданный файл можно открыть самой диалоговой системе.

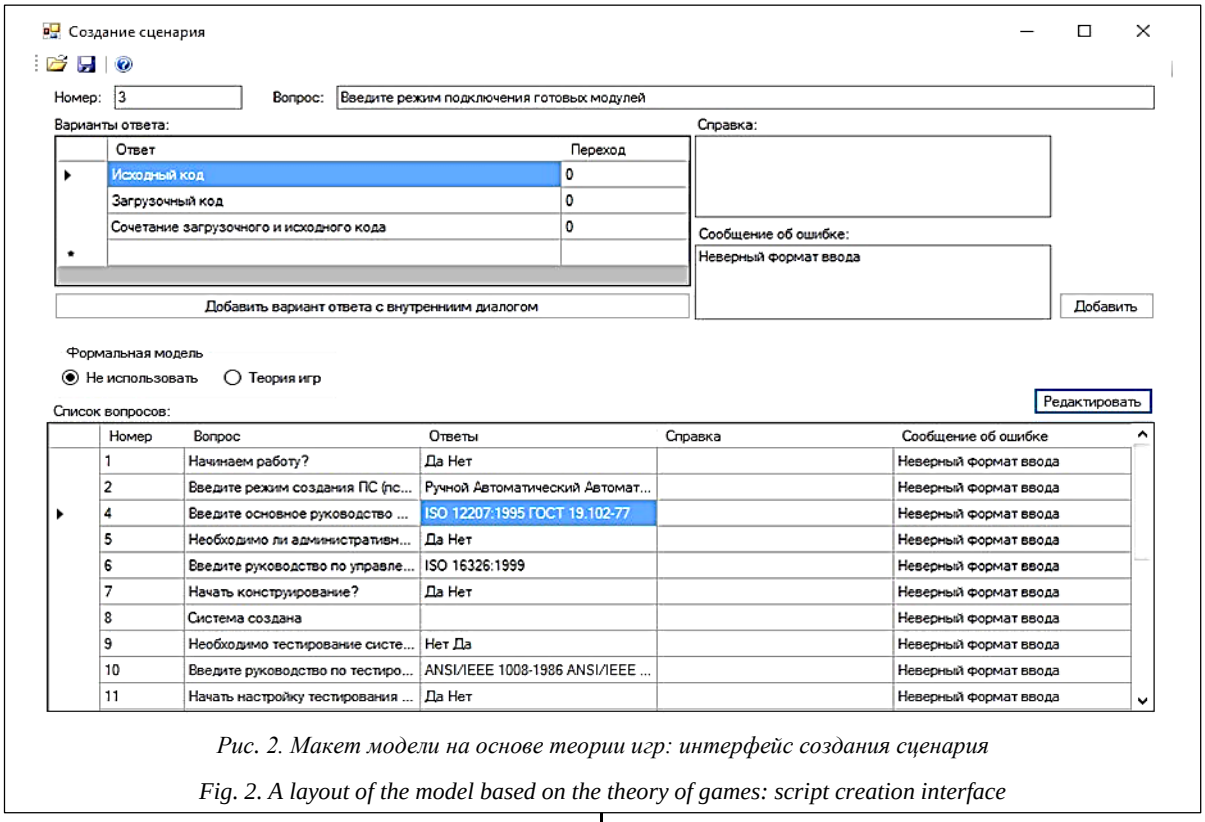

Рис. 2. Макет модели на основе теории игр: интерфейс создания сценария

Fig. 2. A layout of the model based on the theory of games: script creation interface

Пользователю предлагается пройтись по сценарию диалога и ответить на вопросы. Исходя из ответа пользователя система переходит в необходимое состояние и записывает результаты.

Макет на основе графовой модели. Графовая (автоматная) модель является одной из доступных и понятных моделей для реализации программных систем.

Структуру макета (рис. 3) составляют две части: реализация модели с заданными свойствами с помощью так называемого генератора сценария и

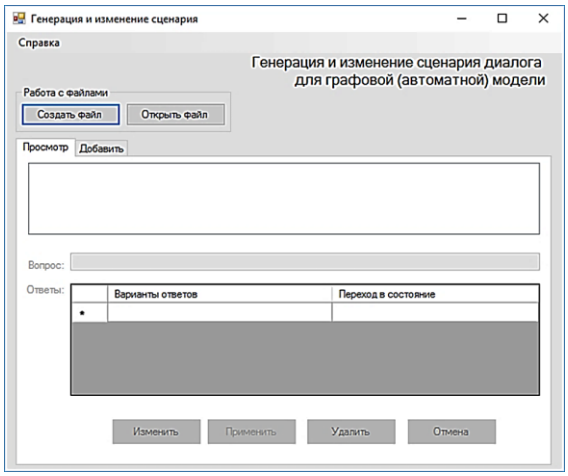

Рис. 3. Макет графовой модели: интерфейс создания сценария

Fig. 3. A layout of the graph model: the script creation interface

применение построенного сценария в диалоговой системе. В качестве прототипной практической реализации графовой (автоматной) модели она будет выглядеть следующим образом:

– каждой вершине графа соответствует вопрос диалога;

– каждому переходу в графе соответствует один из возможных вариантов ответов на вопрос, что по сути дает возможность перемещаться в любые доступные состояния;

– каждому вопросу аналогично вершине графа присваивается некоторое состояние.

Создание сценария, построенного на основе графовой (автоматной) модели, начинается с создания пустого документа в формате .xml. Для записи данных состояния (вершины) графа используется поле «Вопрос», в которое записывается текст вопроса. Граф реализован таблицей с полями «Варианты ответов», «Переход в состояние». После заполнения формы необходимо нажать кнопку «Добавить вопрос». Путем таких последовательных действий с указанием номера перехода на определенное состояние для каждого варианта ответа на вопрос можно сформировать файл, в котором хранится сценарий, построенный на основе модели.

После выполнения этих действий можно открыть заполненный файл сценария в главном окне программы. После таких манипуляций пользователю предлагается пройтись по сценарию диалога и ответить на вопросы. Исходя из ответа пользователя система переходит в необходимое состояние и записывает результат в «Ход диалога». В ходе диа-

лога присутствует анализатор ошибок, который выводит в поле «Сообщение об ошибке» информацию пользователю о том, что он ввел неверный ответ, и система вновь просит его ответить на вопрос, выбрав один из предложенных вариантов ответа.

Макет на основе модели, использующей сетевое представление в виде тензорной сети. Модель, использующая тензорное сетевое представлен, подробно описана в [17], и данное описание взято за основу для создания макета модели. Задача построения информационной системы ставится как задача сети. Имеется сеть, состоящая из совокупности одномерных симплексов. На сеть воздействуют некоторыми величинами. Необходимо определить величины откликов, создаваемых сетью. Задание на сети различных систем координат, переход от одной координатной системы к другой и проецирование инвариантного объекта в различные системы координат дают возможность управлять сетью.

В основу общей теории информационных систем положена возможность непрерывных преобразований, которые позволяют переходить от одних систем к другим. В пассивной интерпретации эти преобразования можно рассматривать как преобразования от одних обобщенных координат к другим, в активной – как преобразование от одной системы к другой. В результате изучение и проектирование сложной информационной системы будет сводиться к изучению простых систем.

Формальное определение сети можно дать так: на счетном множестве M задано бинарное отношение R как подмножество $M \times M$: $R \in M \times M$, $x \in M$, $y \in M$, $(x, y) \in R$, где (x, y) – упорядоченная пара.

Определим преобразование $T(R) \Rightarrow T(r)$.

Оно устанавливает взаимно однозначное соответствие между элементами $r = (x_r, y_r)$ системы и непустым подмножеством одномерных симплексов с концами в точках x_r и y_r. Сетью называется система (множество одномерных симплексов), вершинам и ребрам которой поставлены в соответствие некоторые величины, то есть определены функции: $f_1: M \to x$, $f_2: S \to y$.

Отображения, сохраняющие математические структуры на множествах, позволяют сопоставить между собой различные пространства, найти среди них изоморфные или согласованные в более слабом смысле. Множество всех отображений представляет новое множество, отличное от исходных множеств, его называют функциональным пространством. Использование функционального пространства позволяет строить множества (системы) с нужными свойствами или структурами.

Покажем, как строится обобщенная топологическая модель информационной системы, которая затем, проецируясь в частные системы координат, будет давать конкретные модели и способы перехода от одной системы к другой. Структуру системы задают способы соединения ветвей (одномерных симплексов). В общем случае в качестве

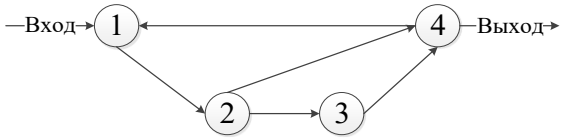

Рис. 4. Схема сети ДС

Fig. 4. A diagram of dialogue system network

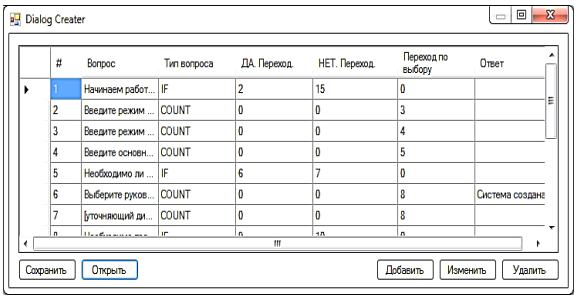

Рис. 5. Макет модели на основе тензорной сети: интерфейс редактора сценариев

Fig. 5. A layout of the model based on the tensor network: the script editor interface

ветвей используются плоскости, n-мерные объемы. Каждая ветвь рассматривается как независимое измерение n-мерного пространства.

Модель информационной системы представляется в общем случае в виде сети, вершинам соответствуют домены (множество данных одного типа), а ветвям – преобразования между данными. Сеть представляет собой структуру, в которой имеются как замкнутые, так и разомкнутые пути. Эта структура, образованная из ветвей и узлов, рассматривается как одномерный комплекс.

В случае ДС узлы сети будут вопросами, а дуги – преобразованиями системы, то есть переходами от одной системы координат в другую (рис. 4). Сеть задается тензором банка данных, который хранит в себе все сущности и их связи. Задание вопроса можно сравнить с вычислением функции, в зависимости от ответа выполняется переход к следующему узлу сети, а если для данного ответа нет функции перехода, то она определяется исходя из топологии сети. Для каждого вопроса нужно определить множество пар <ответ, переход>, если пары не будет, переход определяется из топологии сети.

В ходе исследования был разработан комплекс, состоящий из редактора сценариев и ДС (рис. 5).

Эксперт вводит вопрос системы, настраивает тип ответа от пользователя, выбирает узлы, в которые происходит переход при выборе того или иного варианта и поступает ответное сообщение системы. Все введенные данные проверяются на корректность.

ДС состоит из одной формы, на которой меняются вопросы, также могут появляться всплывающие сообщения, если в них есть необходимость (структуру программы см. на рисунке 6).

Рис. 6. Схема работы программы

Fig. 6. A program operation scheme

Заключение

В работе были рассмотрены концептуальные основы построения генератора ПО сложных систем как сложной системы с макетно-модельным подходом к разработке, позволяющим в короткие сроки получать базовый макет системы, пригодный для дальнейшего итеративного расширения, для тестирования и учебных целей.

Определены концепции построения подсистемы для решения проблемы совместимости программных модулей, входящих в генерируемую систему. Архитектурой, позволяющей сделать систему максимально гибкой и легкой для расширения дополнительной функциональностью, является микроядерная архитектура. Плагины для расширения функциональности разделены на несколько групп, тем самым введено явное соглашение о наборе функций системы, что облегчает ее тестирование.

Сделаны обзор различных моделей человеко-машинного взаимодействия в формате ДС, сравнение их преимуществ и недостатков, для учебно-макетной реализации представлены три модели: сетевая, графовая и модель, основанная на теории игр.

Выводы

Полученные макеты моделей человеко-машинного взаимодействия будут использованы в учебном процессе в КФУ (г. Казань) по курсам «Проектирование и архитектура программных систем», «Объектно-ориентированный анализ и проектирование» и «Конструирование программного обеспечения».

Дальнейшая работа заключается в разработке макетов для остальных моделей, чтобы более полно исследовать их достоинства и недостатки и на их основе выбрать необходимую модель при разработке сборочного генератора программных систем. Особенно выделяется малоизученное направление естественно-языковой модели диалогового взаимодействия, исследование которого представляется важной задачей.

Литература

1. Георгиев В.О. Кодовое представление основных понятий ООА и П в разных языковых средах. Ч. 1. Казань: Изд-во Казан. ун-та, 2016. 230 с.

2. Георгиев В.О. Учебно-модельный вариант интерактивной системы генерации ПО сложных систем с предварительной прединтерпретацией программных модулей // Информационные системы и технологии (ИСТ-2016): матер. XXII Междунар. науч.-технич. конф. Нижний Новгород, 2016. С. 248–249.

3. Георгиев В.О. Программная инженерия: технологические принципы разработки программного обеспечения на уровне формализованного описания на примере абстрагированной диалоговой системы // Языковая семантика: модели и технологии: матер. тр. Междунар. конф. TEL'2014. Казань, 2014. С. 40–41.

4. Georgiev V.O., Prokopiev N.A. Model Approach To Interactive System Software Development. Intern. Jour. of Applied Engineering Research (IJAER). 2015, vol. 10, no. 24, pp. 45208–45213.

5. Поликашин Д.С., Еникеев А.И., Георгиев В.О. Исследование проблем автоматизации решения задачи совместимости программных систем // Информационные системы и технологии (ИСТ-2016): матер. XXII Междунар. науч.-технич. конф. Нижний Новгород, 2016. С. 251.

6. Richards M. Software architecture patterns. Sebastopol: O'Reilly Media, 2015, 47 p.

7. Ponomarenko A., Rubanov V. Automated verification of shared libraries for backward binary compatibility. Proc. 2nd Intern. Conf. on Advances in System Testing and Validation Lifecycle. VALID '10. 2010, pp. 57–62.

8. Ponomarenko A., Rubanov V. A combined technique for automatic detection of backward binary compatibility problems. Proc. 8th Intern. Conf. Perspectives of System Informatics, PSI'11, 2011, pp. 313–321.

9. Würsch M. Improving abstract syntax tree based source code change detection. Zurich, University of Zurich Publ., 2006, 78 p.

10. Саух А.М. Анализ некоторых семантических аспектов исходных текстов программ на основе формальных спецификаций синтаксиса и семантики // Прикладная дискретная математика: Приложение. 2012. № 5. С. 110–111.

11. Fluri B., Wuersch M., Pinzger M., Gall H. Change distilling: tree differencing for fine grained source code change extraction. IEEE Transactions on Soft. Eng., 2007, vol. 33, no. 11, pp. 725–743.

12. Nigay L., Bouchet J., Juras D., Mansoux B., Ortega M., Serrano M., Lawson. J.-Y. L. Software engineering for multimodal interactive systems. Multimodal User Interfaces From Signals to Interaction. 2008, pp. 201–218.

13. Chatty S. Programs = Data + Algorithms + Architecture: consequences for interactive software engineering. Engineering Interactive Systems. 2008, pp. 356–373.

14. Camila Cordero Mansilla, Ángel de Miguel Artal, Eladio Domínguez Murillo, Ma Antonia Zapata Abad. Modelling interactive systems: an architecture guided by communication objects. HCI Related Papers of Interacción 2004, pp. 345–357.

15. Sauer S., Breiner K., Hussmann H., Meixner G., Pleuss A., Van den Bergh J. Combining Design and Engineering of Interactive Systems through Models and Tools (ComDeisMoto). Human-Computer Interaction – INTERACT 2011, pp. 724–725.

16. Maudet. N., Evrard. F. A generic framework for dialogue game impementation. Proc. 2nd Workshop on Formal Semantics & Pragmatics of Dialogue. 1998, pp. 185–198.

17. Арменский А.Е. Тензорные методы построения информационных систем. М.: Наука, 1989. 152 с.

Software & Systems Received 09.08.17
DOI: 10.15827/0236-235X.121.184-191 2018, vol. 31, no. 1, pp. 184–191

CONCEPTUAL BASIS FOR DEVELOPING TRAINIG MODELS OF A COMPLEX SOFTWARE SYSTEM ASSEMBLING GENERATOR

V.O. Georgiev [1, 2], *Ph.D. (Engineering), Expert of the PAS, Senior Lecturer, VOGeorgiev@kpfu.ru*
N.A. Prokopev [2], *Assistant, nikolai.prokopyev@gmail.com*
D.S. Polikashin [2], *Postgraduate Student, nullexception@yandex.ru*

[1] *Academy of Sciences of the Russian Federation, Leninsky Av. 14, Moscow, 119991, Russian Federation*
[2] *Kazan (Volga region) Federal University, Kremlevskaya St. 18, Kazan, 42008, Tatarstan, Russian Federation*

Abstract. The paper presents a conceptual basis for developing training prototypes of an interactive assembling system for automatic building of application software systems. This basis was obtained during practical works in such courses as "Software systems design and architecture" and "Object-oriented analysis and design".

Software systems generation is based on the Theory of program schemes concepts, its standard structural schemes and program engineering methodologies. Assembling occurs in user interaction on a professional language. The professional dialogue language is based on the concept of intellectual interface and comprises a combination of a strictly programmed dialogue and a free scripted dialogue.

The purpose of the system is practical demonstration of basic stages and operations when developing complex software according to a layout-modelling approach to software engineering briefly summarized in the paper.

The paper also includes formulations and solutions for some important problems of complex software systems generation, such as program module compatibility, formalization of computer interaction and choosing formal models for human-machine interface. The authors propose a specific system architecture that allows expanding the system with process plugins when collecting incompatible software modules, which are written in different languages and require diverse libraries, to make them compatible. The paper presents systemized formal and informal models of human-computer interaction. In addition, there are three training prototype implementations for selected models: model based on the game theory, a tensor network model and a graph-based model.

Keywords: software generator, code conversion, training model, interactive system, tools and technologies.

References

1. Georgiev V.O. *Kodovoe predstavlenie osnovnykh ponyaty OOA i P v raznykh yazykovykh sredakh* [Code Representation of Basic Concepts of OOA and P in different Language Environments]. Part 1. Kazan, Kazan Univ. Publ., 2016, 230 p.

2. Georgiev V.O. Education-model version of the interactive software generation system for complex systems, with preliminary preinterpretation of software modules. *Materialy XXII Mezhdunar. nauch.-tekhnich. konf. "Informatsionnye sistemy i tekhnologii (IST-2016)"* [Proc. 12th Int. Science and Tech. Conf. Information Systems and Technologies (IST-2016)]. N. Novgorod, 2016, pp. 248–249 (in Russ.).

3. Georgiev V.O. Software engineering: technological principles of software development at the level of a formalized description, using the example of an abstracted dialog system. *Materialy tr. mezhdunar. konf. TEL'2014 "Yazykovaya semantika: modeli i tekhnologii"* [Proc. Int. Conf. TEL'2014. Language Semantics: Models and Technologies]. Kazan, 2014, pp. 40–41 (in Russ.).

4. Georgiev V.O., Prokopiev N.A. Model Approach to interactive system software development. *Int. Jour. of Applied Engineering Research (IJAER)*. 2015, vol. 10, no. 24, pp. 45208–45213.

5. Polikashin D.S., Enikeev A.I., Georgiev V.O. Investigation of problems of automation of the problem of compatibility of software systems. *Materialy XXII Mezhdunar. nauch.-tekhnich. konf. "Informatsionnye sistemy i tekhnologii (IST-2016)"* [Proc. 12th Int. Science and Tech. Conf. Information Systems and Technologies (IST-2016)]. N. Novgorod, 2016, 251 p. (in Russ.).

6. Richards M. *Software Architecture Patterns*. Sebastopol, O'Reilly Media Publ., 2015, 47 p.

7. Ponomarenko A., Rubanov V. Automated Verification of shared libraries for backward binary compatibility. *VALID '10: Proc. 2010 2nd Int. Conf. on Advances in System Testing and Validation Lifecycle*. 2010, pp. 57–62.

8. Ponomarenko A., Rubanov V. A combined technique for automatic detection of backward binary compatibility problems. *PSI'11: Proc. 8th Int. Conf. on Perspectives of System Informatics*. 2011, pp. 313–321.

9. Würsch M. *Improving Abstract Syntax Tree based Source Code Change Detection*. Zurich, Univ. of Zurich Publ., 2006, 78 p.

10. Saukh A.M. Analysis of some semantic aspects of source code of programs based on formal specifications of syntax and semantics. *Prikladnaya diskretnaya matematika. Prilozhenie* [Applied Discrete Mathematics. Application]. 2012, no. 5, pp. 110–111 (in Russ.).

11. Fluri B., Wuersch M., Pinzger M., Gall H. Change distilling: tree differencing for fine grained source code change extraction. *IEEE Trans. on Software Engineering*. 2007, vol. 33, no. 11, pp. 725–743.

12. Nigay L., Bouchet J., Juras D., Mansoux B., Ortega M., Serrano M., Lawson. J.-Y. L. Software engineering for multimodal interactive systems. *Multimodal User Interfaces from Signals to Interaction*. 2008, pp. 201–218.

13. Chatty S. Programs = Data + Algorithms + Architecture: Consequences for interactive software engineering. *Engineering Interactive Systems*. 2008, pp. 356–373.

14. Camila Cordero Mansilla, ángel de Miguel Artal, Eladio Domínguez Murillo, Ma Antonia Zapata Abad. Modelling interactive systems: an architecture guided by communication objects. *HCI related papers of Interacción 2004*. 2008, pp. 345–357.

15. Sauer S., Breiner K., Hussmann H., Meixner G., Pleuss A., Van den Bergh J. Combining design and engineering of interactive systems through models and tools (ComDeisMoto). *Human-Computer Interaction – INTERACT 2011*. 2011, pp. 724–725.

16. Maudet N., Evrard F. A generic framework for dialogue game impementation. *Proc. 2nd Workshop on Formal Semantics & Pragmatics of Dialogue*. 1998, pp. 185–198.

17. Armensky A. *Tenzornye metody postroeniya informatsionnykh sistem* [Tensor Methods for Information Systems Creation]. Moscow, Nauka Publ., 1989, 152 p.