УДК 004(4'22+054)

МОДЕЛИРОВАНИЕ АРХИТЕКТУРЫ И ПОСЛЕДОВАТЕЛЬНОСТИ ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ И НФОРМАЦИОННОЙ СИСТЕМЫ

ЧАЙНИКОВ С.И., СОЛОДОВНИКОВ А.С.

Предлагается графовая модель архитектуры программного обеспечения информационной системы и автоматная модель проверки выполнения ограничений к нему. Графовая модель описывает архитектуру с избыточной функциональностью и используется для конфигурирования рабочих мест на основе требований конечного пользователя. Выполняемость требований проверяется на базе автоматной модели, которая также используется для управления вычислительными процессами.

1. Обзор существующих методов автоматизированного синтеза программного обеспечения информационной системы

Для процесса проектирования программного обеспечения (ПО) за основу может быть взята одна из существующих технологий проектирования (SADT, IDEF, SSADM, Meris), которые используют однотипные этапы жизненного цикла (ЖЦ) информационных систем (ИС). В рамках ЖЦ для обеспечения автоматизации процесса синтеза необходимо формализованное описание как самой предметной области (ПрО), так и структуры ПО вместе с требованиями к конечному продукту. Использование такого подхода [1] обосновывает создание инструментария – так называемого генератора проектов – на базе совокупности формальных документов, адекватно отражающих ПрО, который позволяет на конечных этапах генерировать программный код системы и выполнять технологическую сборку.

Разработки в данном направлении велись с 80-х годов [2], при этом методы имеют различные недостатки: например, использование генерируемых скриптов вместе с текстами программного кода для сборки программ проекта по исходным текстам в соответствии с выбранной платформой, в связи с чем необходимо дополнительное ПО, анализирующее полученные скрипты. Дополнительное ПО (или программные инструментарии), позволяющее осуществлять синтез программы, может работать в диалоговом режиме и быть основанным на: 1) запоминании состояния задачи на любом этапе в целях восстановления ее состояния до заданного момента времени; 2) проверке исходных данных задачи до ее решения, в процессе решения и после него; 3) оперативном вводе исправлений в исходных данных; 4) предоставлении пользователю возможности многосеансной работы. Указанные особенности являются преимуществом и позволяют оптимизировать процессы синтеза. Такой функционал позволяет пользователю прорабатывать различные стратегии решения задачи, минимизировать время ожидания решения задачи за счет снижения количества ошибок, приводящих к сбою вычислительного процесса (ВП), и разбивать последовательность действий пользователя на этапы (сеансы) с длительными временными перерывами между ними. Однако для осуществления синтеза программного продукта с помощью подобных программных инструментариев требуется развитие проблемно-ориентированного языка.

В рамках использования диалоговых систем автоматизированного синтеза ПО и построения ВП обычно используются графовые модели вычислений [3, 4], которые позволяют формировать ВП на основе маршрутов, выделяемых на графе. В дальнейшем развитие диалоговых систем обозначилось в сторону проектирования диалога на базе естественного языка и получило развитие путем применения искусственного интеллекта и баз знаний.

На современном этапе существует также подход к автоматизированной генерации ПО, основывающийся на использовании алгебраических спецификаций – системах алгоритмических алгебр (САА) В.М. Глушкова [5] и методе диалогового конструирования синтаксически правильных программ [6]. Особенность заключается в совместном использовании трех представлений алгоритма при его конструировании: аналитического (САА-формула), естественно-лингвистического и графового (граф-схемы). Такое направление получило развитие в представлении алгоритмов в виде САА-М-схем со смещением акцента в сторону разработки инструментария автоматизированного преобразования параллельных алгоритмов [7] с использованием технологий MPI (Message Passing Interface) и многопоточности в Java. Недостатком этих методов является то, что их реализация зависит от конкретного языка программирования.

Выделяют также технологию автоматического синтеза ПО с использованием онтологии прецедентов [8], в которой прецеденты определяются как пара <S, P>, где S — спецификация и P – соответствующая ей программа. Такое онтологическое описание позволяет накапливать опыт разработки, выполнять автоматическую классификацию программ на основе их спецификаций и построение программ путем адаптации известных решений.

Другое направление в автоматическом синтезе ПО получило развитие с разработкой технологии генетических алгоритмов. Они позволяют определять структуру управляющего автомата, который в свою очередь является системой вложенных и взаимовызываемых автоматов [9]. Такой подход реализует технологию автоматного программирования и обладает такими преимуществами как автоматизация процесса верификации, документируемости, упрощение процедуры внесения изменений.

Автоматный подход широко применяется не только в синтезе программного кода [10], но и в управлении поведением самой системы, запущенной на выполнение [11].

Использование архитектурных шаблонов и проблемно-ориентированных языков описания архитектуры применяется в развитии технологии построения композитных приложений [12, 13]. Примером служит инструментально-технологическая платформа CLAVIRE, использующая проблемно-ориентированный язык EasyFlow и позволяющая автоматизировать формирование архитектурных шаблонов, на основе которых выполняется генерация сценариев запуска приложения в распределенной среде [14].

С усложнением предметных областей увеличивается сложность аппаратного обеспечения и ПО. Повышаются требования к эффективности и производительности ПО (стандарт ISO/IEC 25041:2012). Пока ПО по скорости развития отстает от аппаратного, поэтому требует использования особых технологий по оптимизации своей работы.

Сервисно-ориентированные технологии организации кластерных вычислений являются на данный момент наиболее перпективными. Они порождают новое ответвление – так называемые облачные технологии. Однако в данном направлении присутствует существенная проблема – обеспечение безопасности данных, находящихся во владении сторонних организаций. К известным типам угроз (сетевые атаки, вредоносное ПО, уязвимости в приложениях и ОС) при использовании облачных технологий добавляются сложности, связанные с контролем среды (гипервизора), трафика между гостевыми машинами и разграничением прав доступа [15].

2.Архитектура программного инструментария для автоматизированного синтеза программного обеспечения

Каждое из данных направлений требует своих подходов, эффективных для определенного класса задач и архитектур вычислительных систем, а также детального изучения и развития. Поэтому современные технологии проектирования ПО должны быть сориетированы на перспективу развития аппаратного обеспечения и иметь возможность адаптации их принципов к многопроцессорным и распределенным архитектурам.

На основе приведенного анализа можно выделить следующие методы синтеза ПО ИС:

1) ручной;

2) автоматизированный (в том числе с использованием метода диалогового конструирования);

3) автоматический.

При этом в рамках автоматизированного и автоматического методов наиболее широко используются следующие средства формализации:

1) логико-алгебраические спецификации;

2) автоматные модели;

3) графовые модели;

4) ADL-языки;

5) онтологические средства.

Перечисленные средства формализации могут применяться в рамках синтезирующего (на базе модели вычислений, отображающей понятия и отношения ПрО и программной спецификации), композиционного (на базе функций и операций композиции в логико-математической системе) и сборочного программирования (на базе модели сборки в виде ориентированного нагруженного графа) [12, 13].

Что касается методов проектирования и разработки ИС с использованием наиболее распространенного сборочного подхода, на данный момент существует следующая обобщающая классификация этих методов [13]:

1) модульно-ориентированный;

2) объектно-ориентированный;

3) компонентно-ориентированный;

4) метод генерации;

5) сервисно-ориентированный.

Каждый следующий метод является развитием предыдущего. Реализация этих методов основывается на использования хранилищ готовых решений, компонент повторного использования, а также на наличие существенных проблем – обеспечение межмодульного интерфейса при сборке ИС [13] и существование конфликта между нефункциональными требованиями к компонентам.

В целях повышения эффективности синтеза ПО ИС с помощью CASE-средств и предоставления возможности управления действиями программы указывается на необходимость разделения программной системы на управляющий объект и объект управления. Управляющим объектом может служить некая исполнительная система или диспетчер. В большинстве случаев в составе инструментальных средств, позволяющих обеспечивать автоматизацию процессов сборки ПО, определяют компоновщик, формирующий общий программный код на основе атомарных фрагментов и алгоритма сборки, и так называемый диспетчер, который получается на выходе процесса сборки наряду с целевой ИС (рис. 1) [16].

Такой диспетчер позволяет управлять функционированием данной системы. Кроме того, при указанной архитектуре компоновщик может быть сориентирован на повторное использование программных модулей, компонентов, ПО. Может применять версионность сборок, а архитектура ИС может быть расширена до включения в состав нескольких компаний-разработчиков ПО, что влечет за собой необходимость разработки процессов повторного применения

программных компонентов и механизмов управления этими процессами.

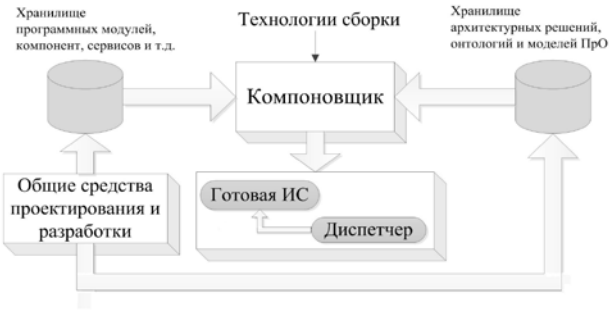

Рис. 1. Концептуальная структура инструментальных средств, реализующих сборку ИС

В основу работы компоновщика взята графовая модель архитектуры ПО ИС, которая содержит информацию о правилах взаимодействия программных модулей между собой. Работа над формированием этой графовой модели и вводом соответствующей информации осуществляется с помощью разработанного программного инструментария, поддерживающего разработчика в рамках процессов анализа требований к программным средствам, проектирования архитектуры, конструирования и комплексирования программных средств ЖЦ, создания ПО. При этом проверка корректности стыковки программных модулей и управление их взаимодействием ложится на автоматную модель ВП, которая синтезируется при помощи автоматного метода проверки выполнения ограничений к формируемому ПО. Разработчик в визуальной или в табличной форме описывает архитектуру будущего ПО с помощью графовой модели, на основе которой формируется автоматная модель ограничений формируемого ПО. Конечные автоматы (КА) данной модели при помощи архитектурных шаблонов реализуются в виде классов управляющих автоматов на базе объектно-ориентированных языков программирования и позволяют управлять реализацией ВП программы. Зададим формально информацию, получаемую из графовой модели архитектуры ПО ИС.

3. Модель архитектуры программного обеспечения информационной системы

Упомянутые инструментарии реализуют CASE-технологии, поддерживающие и автоматизирующую работы на стадиях ЖЦ, тем самым ускоряя процессы проектирования и разработки систем и повышая их качество. CASE-средства обладают достаточно большими преимуществами [17]: качество разрабатываемого ПО за счет средств автоматического контроля и генерации, повышают уровень технологической поддержки процессов разработки и сопровождения ПО, улучшают производительность и качество продукции при соблюдении стандартов и документирования. При использовании CASE-средств обнаруживают высокий уровень отдачи от инвестиций в инструментарий, возможность повторного использования компонентов разработки и поддержку адаптивности и сопровождения ПО. CASE-средства позволяют снизить время создания системы, получая ее прототип и оценку на ранних стадиях проектирования. Позволяют осуществлять коллективную разработку ПО в режиме реального времени.

Данные достоинства характеризуют CASE-средства как высокоэффективный инструмент. Тем не менее, в зависимости от специфики и сложности конкретной ПрО всегда требуются устовершенствования методов, заложенных в этот инструментарий.

Автоматизация процесса синтеза с помощью CASE-средств базируется на упорядоченной и структурированной информации о необходимых программных библиотеках, участвующих в формировании программы, об автоматных моделях, встраиваемых в структуру формируемого ПО, о формализованном представлении архитектуры ПО в виде графовой модели и программных спецификаций, регламентирующих правила стыковки программных модулей и функционирование компоновщика ПО. Все эти данные должны сохраняться на каждой стадии проектирования и разработки ПО, чем обосновано использование специальных форматов данных, обеспечивающих хранение и использование необходимой информации для разработчика в файле проекта. Такая информация необходима для того, чтобы обеспечить взаимосвязать программных компонентов, управляющего КА и графовой модели. Учитывая, что разработчику, который использует программный инструментарий автоматизированного синтеза ПО, требуется визуально отображать данные графовой модели на экранных формах, а также обеспечить взаимосвязь данных графовой модели архитектуры ПО и автоматной модели проверки выполнения ограничений, определим формально кортеж G_{SP}. Этот кортеж специфицирует данные графовой модели и структуру файла проекта:

$$G_{SP} = \langle N,S,VC,AM,CG,D \rangle , \qquad (1)$$

где N – множество координат вершин, размещаемых на рабочей области экранной формы; S – множество стадий проекта, фиксирующих текущее состояние процесса компиляции ПО; VC – множество характеристик вершин графовой модели, описывающих наличие компонент сильной связности; AM – множество атрибутов, определяющих взаимосвязь вершин в соответствии с матрицей смежности графовой модели; CG – множество атрибутов, определяющих взаимосвязь между вершинами в соответствии с матрицей смежности конденсированного графа; D – множество атрибутов, определяющих характеристики функций, которые сопоставляются с вершинами графовой модели в соответствии с отображением «вершина-функция».

Для кортежа (1) множество координат вершин определяется следующим образом:

$$N = \left\{ m_1^N, m_2^N, ..., m_k^N \right\}, \; m_i^N = \{vx, vy\} , \qquad (2)$$

где vx, vy – координаты i-й вершины. Стадии проекта определяются как

$$S=\{s_1,s_2,...,s_r\},\ s_i=\{\text{True,False}\}\ , \quad (3)$$

где s_i – логическая переменная, обозначающая текущую активную стадию проекта. Это может быть, например, стадия, на которой осуществляется ввод матрицы смежности графовой модели или стадия проверки корректности введенной информации.

Множество характеристик вершин графовой модели определяется так:

$$VC=\{m_1^{VC},m_2^{VC},...,m_n^{VC}\}, \quad (4)$$

здесь $m_i^{VC}=\{\text{isSubG,Comp}\}$ – подмножество, состоящее из двух элементов: *isSubG* – логическая пременная, которая определяет наличие вложенного подграфа для супервершины (т.е. той вершины, которая образовалась после процесса конденсации и заменила собой часть графа $G=\langle V,X\rangle$; подмножество Comp – перечисляет номера вершин, принадлежащих компоненте сильной связности. Если задать $G_i^{SC}=\langle V^{SC},X^{SC}\rangle$ (где V^{SC} – множество вершин, а X^{SC} – множество ориентированных дуг) как *i*-ю компоненту сильной связности $G_i^{SC}\subset G$, обнаруживаемую при одноименном поиске, то тогда множество *Comp* будет совпадать с множеством V^{SC}, т.е. Comp= V^{SC}, где $V^{SC}\subset V$.

Множество атрибутов, позволяющих задать матрицу смежности ориентированного графа в списочном виде, определяется как

$$AM=\{m_1^{AM},m_2^{AM},...,m_k^{AM}\}, \quad (5)$$

где каждый атрибут $m_i^{AM}=\{V_{adj}\}$ представляет собой список вешин $v_j\in V_{adj}$, в которые идет дуга $x_{ij}\in X$ из вершины $v_i\in V$, при этом $V_{adj}\subset V$.

Матрица смежности конденсированного графа задается в списочном виде с помощью множества

$$CG=\{m_1^{CG},m_2^{CG},...,m_p^{CG}\}, \quad (6)$$

где каждый атрибут m_i^{CG}, по аналогии с (5), содержит список вершин $v_j\in V_{adj}^{CG}$, в которые идет дуга из вершины $v_i\in V$, но, в отличие от (5), дополняется подмножеством состояний ВП ST^{CG}, соответствующего i-й вершине, и подмножеством характеристик архивных данных AR^{CG}. Элемент множества CG определяется таким образом:

$$m_i^{CG}=\{V_{adj}^{CG},ST^{CG},AR^{CG}\}, \quad (7)$$

где $V_{adj}^{CG}\subset V$, $ST^{CG}=\{0,1,2\}$, $AR^{CG}=\{0,1,2\}$.

Другими словами, если задать переменную e^{st}, которая принимает значения из подмножества ST^{CG}, и переменную e^{ar}, принимающую значения из подмножества AR^{CG}, то

$$e^{st}=\begin{cases}0,\text{ если ВП не запущен;}\\ 1,\text{ если ВП запущен;}\\ 2,\text{ если ВП выполнен.}\end{cases}$$

и также

$$e^{ar}=\begin{cases}0,\text{ если архив отсутствует;}\\ 1,\text{ если запущен процесс архивации;}\\ 2,\text{ если архив существует.}\end{cases}$$

Разделение информации об исходном и конденсированном графе продиктовано необходимостью сохранять информацию о слоях графовой модели и обеспечивать быстрый доступ к нужному уровню.

Множество функций, сопоставляемых с вершинами, определяется таким образом:

$$D=\{p_1^D,p_2^D,...,p_q^D\}, \quad (8)$$

где

$$p_i^D=\{\text{UID, PName, PMName, PUPath, PDes, SPath}\} \quad (9)$$

– подмножество элементов, необходимых для описания программных компонент или модулей, которые эти функции выполняют. Для этого определяем UID как уникальный идентификатор ВП; PName – имя процесса, используемое компоновщиком, при определении логической связи между вершиной графовой модели и программным модулем; PMName – реальное имя программного модуля (или библиотеки); PUPath – физический путь к файлу программного модуля; SPath – физический путь к файлу спецификации, которая определяет основные характеристики модуля и является ссылкой на спецификацию модуля Sp_i; PDes – дополнительные сведения об элементе. По сути, атрибут PName является синтаксическим уникальным именем программного модуля, а PMName – реальным физическим именем (семантическим) процесса или модуля, на которое могут ссылаться несколько синтаксических имен.

Разработчик, описывая архитектуру ПО, сопоставляет вершины графовой модели архитектуры ПО ИС с программными модулями, которые выполняют свои функции, и имеет дело прежде всего с атрибутами PName, PMName, PUPath и SPath. Уникальное значение атрибута UID формируется автоматически, а присваивание значения атрибуту PDes является опциональным, т.е., в том случае, если требуются дополнительные данные по программному модулю. Значения PName и PMName теоретически могут совпадать, но на практике это не будет правильным подходом, поскольку очень часто можно найти однотипные функ-

ции для различных подсистем ПО, которые могут быть выполнены одним программным модулем, хранят один экземпляр модуля, используя в других случаях ссылки на него, как это общепринято для стандартной практики использования динамических или статических библиотек. Однако применение различных имен (синтаксических) для одного и того же программного модуля при формировании кортежа p_i^D (9) продиктовано желанием упростить представление и обработку графовой модели, уменьшив количество связей между вершинами, чрезмерное количество которых могло бы привести к появлению дополнительных компонент сильной связности, а значит и супервершин.

Документ, предоставляющий проектировщику необходимую информацию для синтеза ПО, может быть согласован по формату файла с форматом одного из XML-стандартов (например, языка GraphML).

Используя объектно-ориентированный подход, задается принцип взаимодействия программных модулей. Этот принцип базируется на интерфейсах класса, который содержит сигнатуры его компонентов (запросов и команд), а также семантические свойства: предусловия и постусловия запросов и команд, инварианты класса. Для объектно-ориентированного подхода предлагается использовать шаблоны КА (автомата Мили) при разработке управляющего программного модуля (диспетчера) (рис. 2) [18].

Беря во внимание шаблоны проектирования КА и заявленные требования к исполнителю для объектно-ориентированных программ, зададим автоматную модель проверки выполнения ограничений к формируемому ПО.

Для этого примем допущение, что V – является множеством вершин подграфа G, который определяется для текущей задачи, запускаемой пользователем. Обозначим текущий номер яруса буквой L; Imax(L) – количество вершин на ярусе L; Lmax(G) – количество ярусов для подграфа G. Обозначим вычислительный процесс, соответствующий вершине v_i, как P(Vi). Обозначим вершину $v_i \in V$, которая находится на ярусе L, как Vi(V, L) . Также обозначим количество входных дуг (полустепень захода) для вершины $v_i \in V$ как DEG_IN(Vi). А вершину v_k, из которой существует направленная дуга (v_k, v_i), как Vk. На этом основании зададим автомат AForwardSM (рис.3).

ЛОМеЛОМеДля диаграммы автомата AForwardSM (см.рис.3) заданы распараллеливающие и синхронизирующие переходы (табл. 1), обеспечивающие ввод автомата одновременно в несколько состояний (табл. 2). Основные задачи, которые возложены на описанный КА, это:

1) формирование подграфа задачи;

2) проверка состояния задачи;

3) проверка результата на корректность;

4) изменение состояния задачи.

Объектом управления для автомата AForwardSM является программный модуль ForwardUnit, контролирующий прохождение вычислительного процесса по подграфу.

4. Выводы

Графовая модель архитектуры ПО ИС, содержащая информацию, необходимую для формирования архитектуры ПО с избыточной для заданной ПрО функциональностью, позволяет получить гибкую конфигурацию системы в условиях эволюционно изменяющихся требований. Модель позволяет оценить его архитектуру до непосредственного построения и обеспечить динамическое конфигурирование рабочих мест пользователей, что является ее преимуществом. Также предложена автоматная модель проверки выполнения ограничений к формируемому ПО. Модель используется для проектирования исполнителя ВП, моделируя сценарии взаимодействия программных модулей и сравнивая результаты моделирования с нефункциональными требованиями к ПО.

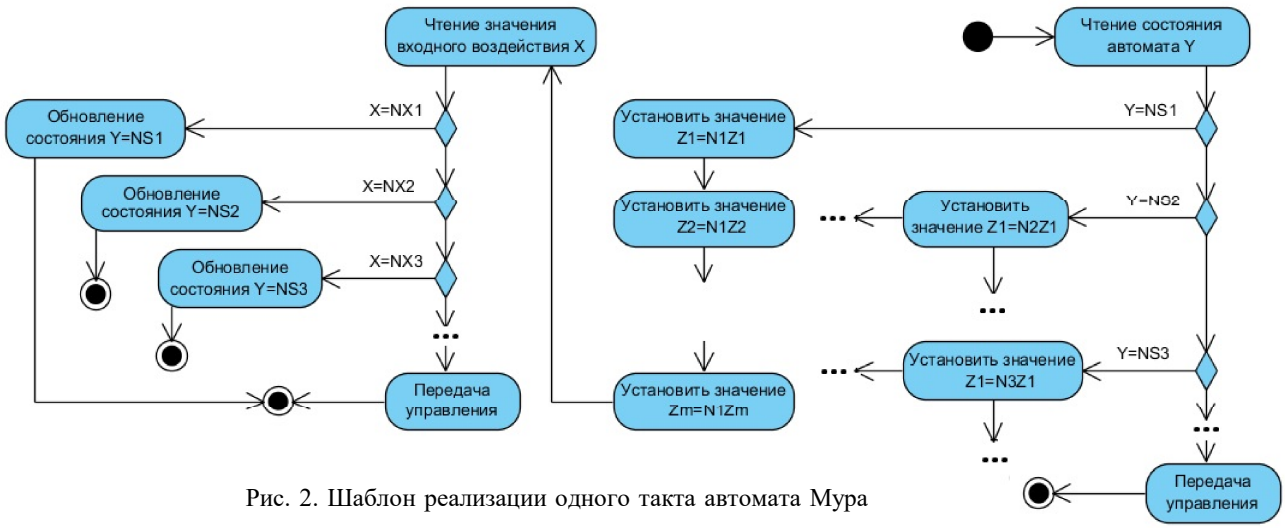

Рис. 2. Шаблон реализации одного такта автомата Мура

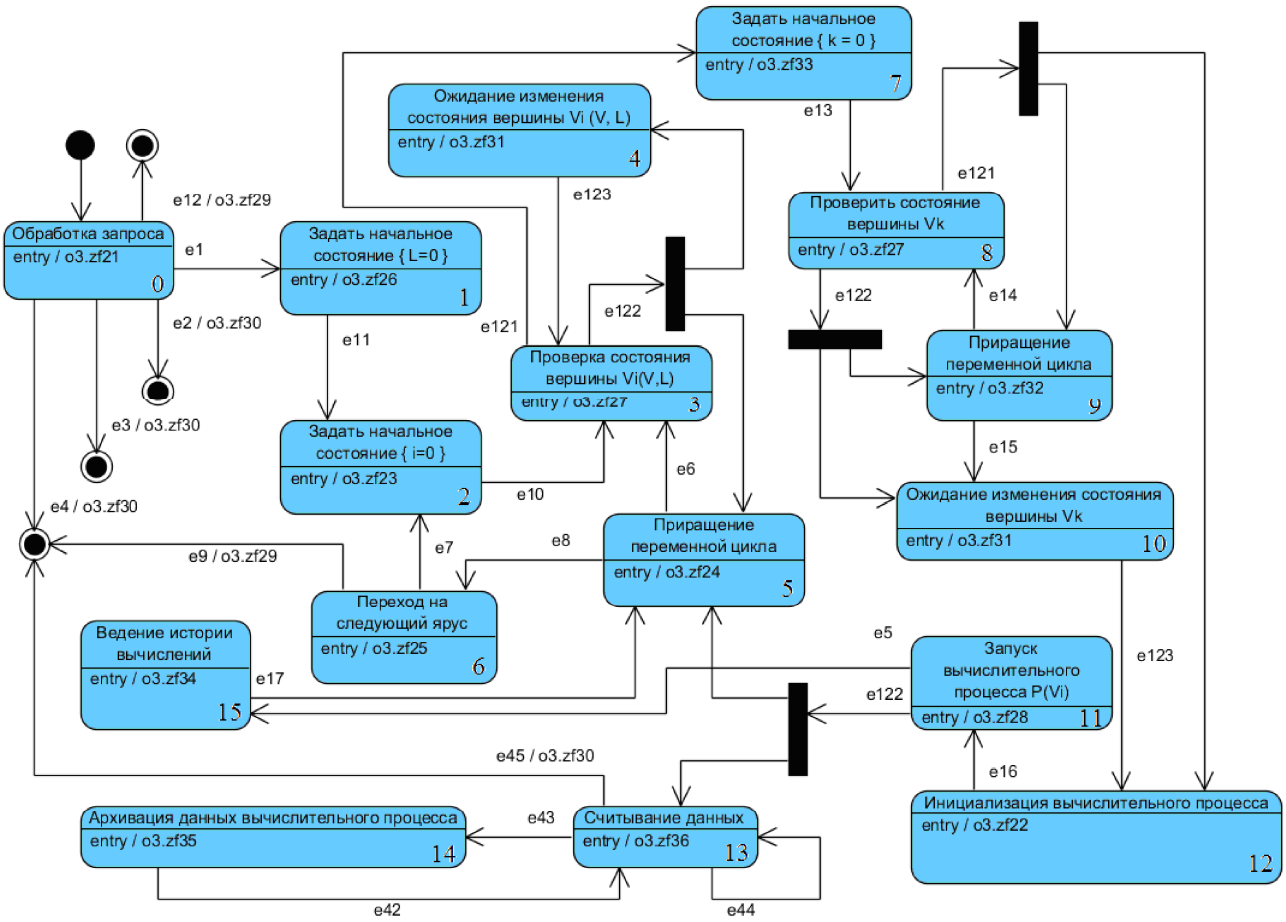

Рис. 3. Диаграмма переходов для вложенного конечного автомата AForwardSM

Таблица 1

Переходы конечного автомата

Обозначение	Наименование
e1/ e2/ e3	Получено разрешение/ Отмена действия/ Действие выполнено
e4/ e5	Процедура выполняется /Ошибка выполнения
e6	Номер вершины меньше максимального для текущего яруса
e7	Номер текущего яруса меньше количества ярусов
e8	Вершина принадлежит множесвту вершин текущего яруса
e9	Достигнут послений ярус ЯПФ подграфа задачи
e10/ e11	Возврат к исходному ВП/ Возврат к исходному ярусу
e12	Сформирован ответ на запрос пользователя
e13	Получено значение полустепени захода для текущей вершины
e14	Обработка входных данных текущей вершины
e15	Множество входящих дуг исчерпано для данной вершины
e16/ e43	Заданы условия и параметры ВП/ Данные считаны
e44/ e45	Время ожидания не превышено / истекло
e121/e122/e123	ВП не запущен / выполняется / выполнен

Таблица 2

Состояния конечного автомата

Обозначение	Наименование
o3.zf21	Инициализациия подграфа вычислительного процесса
o3.zf22	Инициализация ВП для автомата AForwardSM
o3.zf23	Инициализация переменных цикла
o3.zf24 / o3.zf25	Переход к следующему ВП / ярусу ЯПФ подграфа задачи
o3.zf26	Возвращение на исходный ярус ЯПФ подграфа задачи
o3.zf27 /o3.zf28/o3.zf31	Проверка состояния ВП/ запуск ВП/ ожидание ВП
o3.zf32 /o3.zf33	Переход к следующему / исходному ВП
o3.zf34	Сохранение log-записи
o3.zf35 / o3.zf36	Архивация данных/получение данных

Литература: 1. *Генератор* проектов – средство автоматизации проектирования прикладных информационно-вычислительных систем / Ю.А. Флёров, Л.Л. Вышинский, И.Л. Гринёв, А.А. Логинов, и др. // Автоматизация проектирования инженерных и финансовых информационных систем средствами «Генератора проектов». М.: ВЦ РАН. 2010. С. 3-15. **2.** *Инструментальные* средства САПР / Л.Л. Вышинский, И.Л. Гринёв, В.И. Шиленко, Н.И. Широков / / Задачи и методы автоматизированного проектирования в авиастроении. М.: ВЦ АН СССР. 1991. С.52-70. **3.** *Перевозчикова О.Л.* Диалоговые системы / О.Л. Перевозчикова, Е.Л Ющенко. Ин-т кибернетики им. В.М. Глушкова. К.: Наук. думка, 1990. 184 с. **4**. *Jiannong C.* GOP: A graph-oriented programming model for parallel and distributed systems / Cao Jiannong, Alvin T.S. Chan, Yudong Sun // Parallel and distributed computing. 2005. P.21-36. **5**. *Дорошенко Е.А.* Формализованное проектирование эффективных многопоточных программ / Е.А. Дорошенко, К.А. Жереб, Е.А. Яценко // Проблемы программирования. 2007. № 1. С. 17–30. **6**. *Алгеброалгоритмические* модели и методы параллельного программирования / Ф.И. Андон, Е.А. Дорошенко, Г.Е Цейтлин., Е.А. Яценко. Киев: Академпериодика, 2007. 631 с. **7**. *Шкулипа И. Ю.* Автоматизированный синтез программ на основе САА-М-схем / И. Ю. Шкулипа, С. Д. Погорелый // УСиМ. 2010. №4. С. 58-63. **8**. *Корухова Ю. С.* Автоматический синтез программ с использованием онтологии прецедентов / Ю. С. Корухова, Н. Н. Фастовец // Программные системы и инструменты: тематический сборник. 2011.Т. 12. С.203-215. **9**. *Автоматический* синтез системы управления мобильным роботом для решения задачи «Кегельринг» / С.А. Алексеев, А.И. Калиниченко, В.О. Клебан, А.А. Шалыто // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011. № 2 (72). С. 26-31. **10.** *Канжелев С. Ю.* Автоматическая генерация кода программ с явным выделением состояний / С.Ю. Канжелев // Мат. конф.“Software Engineering Conference (Russia) – 2006” (SEC (R). 2006. С. 60 – 63. **11.** *Салмре И.* Программирование мобильных устройств на платформе .NET Compact Framework / Иво Салмре, пер. с англ. М.: Изд. дом «Вильямс». 2006. 736 с. **12.** *Лаврищева Е.М.* Сборочное программирование. Основы индустрии программных продуктов / Е.М. Лаврищева, В.Н. Грищенко.Киев: Наук. думка, 2009. 372с. **13.** *Лаврищева Е.М.* Software Engineering компьютерных систем. Парадигмы, технологии и CASE-средства программирования / Е. М. Лаврищева. К.: Наук. думка. 2013. 283 с. **14**. *Интеллектуальные* технологии распределенных вычислений для моделирования сложных систем / Марьин С. В., Ларченко А. В., Ковальчук С. В., Князьков К. В., и др. // Науч.- техн. вестн. СПбГУ ИТМО. 2010. Вып. № 70. С. 123–124. **15.** *Сергеев Ю.* Управление доступом к виртуальной инфраструктуре с помощью продукта HyTrust / Ю. Сергеев // Jet Info Информационный бюллетень. 2012. №3 (224). 44 с. **16.** *Малышкин В. Э.* Параллельное программирование мультикомпьютеров / В.Э. Малышкин, В.Д. Корнеев. Новосибирск: Новосибирский государственный технический университет. 2006. 452 с. **17.** *Калянов Г. Н.* CASE-технологии: консалтинг в автоматизации бизнес-процессов / Г. Н. Калянов. М.: Горячая линия-Телеком. 2002. 320 с. **18.** *Поликарпова Н. И.* Автоматное программирование / Н. И. Поликарпова, А. А. Шалыто. СПб.: «Питер». 2008. 167 с.

Transliterated bibliography:

1. *Generator* proektov – sredstvo avtomatizatsii proektirovaniya prikladnyih informatsionno-vyichislitelnyih sistem / Yu.A. FlYorov, L.L. Vyishinskiy, I.L. GrinYov, A.A. Loginov, i dr. // Avtomatizatsiya proektirovaniya inzhenernyih i finansovyih informatsionnyih sistem sredstvami «Generatora proektov». M.: VTs RAN. 2010. S. 3-15.

2. *Instrumentalnyie* sredstva SAPR / L.L. Vyishinskiy, I.L. GrinYov, V.I. Shilenko, N.I. Shirokov // Zadachi i metodyi avtomatizirovannogo proektirovaniya v aviastroenii. M.: VTs AN SSSR. 1991. S.52-70.

3. *Perevozchikova O.L.* Dialogovyie sistemyi / O.L. Perevozchikova, E.L Yuschenko. In-t kibernetiki im. V.M. Glushkova. K.: Nauk. dumka, 1990. 184 s.

4. *Jiannong C.* GOP: A graph-oriented programming model for parallel and distributed systems / Cao Jiannong, Alvin T.S. Chan, Yudong Sun // Parallel and distributed computing. 2005. P.21-36.

5. *Doroshenko E.A.* Formalizovannoe proektirovanie effektivnyih mnogopotochnyih programm / E.A. Doroshenko, K.A. Zhereb, E.A. Yatsenko // Problemyi programmirovaniya. 2007. # 1. S. 17–30.

6. *Algebroalgoritmicheskie* modeli i metodyi parallelnogo programmirovaniya / F.I. Andon, E.A. Doroshenko, G.E Tseytlin., E.A. Yatsenko. Kiev: Akademperiodika, 2007. 631 s.

7. *Shkulipa I. Yu.* Avtomatizirovannyiy sintez programm na osnove SAA-M-shem / I. Yu. Shkulipa, S. D. Pogorelyiy // USiM. 2010. #4. S. 58-63.

8. *Koruhova Yu. S.* Avtomaticheskiy sintez programm s ispolzovaniem ontologii pretsedentov / Yu. S. Koruhova, N. N. Fastovets // Programmnyie sistemyi i instrumentyi: tematicheskiy sbornik. 2011.T. 12. S.203-215.

9. *Avtomaticheskiy* sintez sistemyi upravleniya mobilnyim robotom dlya resheniya zadachi «Kegelring» / S.A. Alekseev, A.I. Kalinichenko, V.O. Kleban, A.A. Shalyito // Nauchno-tehnicheskiy vestnik Sankt-Peterburgskogo gosudarstvennogo universiteta informatsionnyih tehnologiy, mehaniki i optiki. 2011. # 2 (72). S. 26-31.

10. *Kanzhelev S. Yu.* Avtomaticheskaya generatsiya koda programm s yavnyim vyideleniem sostoyaniy / S.Yu. Kanzhelev // Mat. konf.“Software Engineering Conference (Russia) – 2006” (SEC (R). 2006. S. 60 – 63.

11. *Salmre I.* Programmirovanie mobilnyih ustroystv na platforme .NET Compact Framework / Ivo Salmre, per. s angl. M.: Izd. dom «Vilyams». 2006. 736 s.

12. *Lavrischeva E.M.* Sborochnoe programmirovanie. Osnovyi industrii programmnyih produktov / E.M. Lavrischeva, V.N. Grischenko.Kiev: Nauk. dumka, 2009. 372s.

13. *Lavrischeva E.M.* Software Engineering kompyuternyih sistem. Paradigmyi, tehnologii i CASE-sredstva programmirovaniya / E. M. Lavrischeva. K.: Nauk. dumka. 2013. 283 s.

14. *Intellektualnyie* tehnologii raspredelennyih vyichisleniy dlya modelirovaniya slozhnyih sistem / Marin S. V., Larchenko A. V., Kovalchuk S. V., Knyazkov K. V., i dr. // Nauch.- tehn. vestn. SPbGU ITMO. 2010. Vyip. # 70. S. 123–124.

15. *Sergeev Yu.* Upravlenie dostupom k virtualnoy infrastrukture s pomoschyu produkta HyTrust / Yu. Sergeev // Jet Info Informatsionnyiy byulleten. 2012. #3 (224). 44 s.

16. *Malyishkin V.E.* Parallelnoe programmirovanie multikompyuterov / V.E. Malyishkin, V.D. Korneev. Novosibirsk: Novosibirskiy gosudarstvennyiy tehnicheskiy universitet. 2006. 452 s.

17. *Kalyanov G. N.* CASE-tehnologii: konsalting v avtomatizatsii biznes-protsessov / G. N. Kalyanov. M.: Goryachaya liniya-Telekom. 2002. 320 s.

18. *Polikarpova N. I.* Avtomatnoe programmirovanie / N. I. Polikarpova, A. A. Shalyito. SPb.: «Piter». 2008. 167 s.

Поступила в редколлегию 10.05.2016

Рецензент: д-р техн. наук, проф. Гребенник И.В.

Чайников Сергей Иванович, канд. техн. наук, профессор кафедры системотехники, Харьковский национальный университет радиоэлектроники. Адрес: Украина, 61166, Харьков, пр. Науки, 14, E-mail: chajnikov@kture.kharkov.ua.

Солодовников Андрей Сергеевич, ассистент кафедры медицинской и биологической физики и медицинской информатики, Харьковский национальный медицинский университет. Адрес: Украина, 61166, Харьков, пр. Науки, 14, E-mail: andrey.sldv@rambler.ru.