OSTIS-2013
(Open Semantic Technologies for Intelligent Systems)

УДК 004.822:514

ПРОГРАММНОЕ УПРАВЛЕНИЕ ПОТОКАМИ РАБОТ В КОНЦЕПТУАЛЬНОМ ПРОЕКТИРОВАНИИ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

Соснин П.И.[*], Лапшов Ю.А.[*], Маклаев В.А.[**]

[*] *Ульяновский государственный технический университет, г. Ульяновск, Россия*
sosnin@ulstu.ru
y.lapshov@ulstu.ru

[**] *Федеральный Научно-Производственный Центр ОАО «НПО «МАРС», г. Ульяновск, Россия*
mars@mv.ru

Представляются средства и язык псевдо-кодового программирования потоков работ, предназначенные для алгоритмического представления бизнес-процессов, исполняемых в проектировании автоматизированных систем (АС). К специфике языка относится его ориентация на интеллектуальный процессор (роль проектировщика), разделяющий общую работу с компьютерным процессором и с другими исполнителями бизнес-процессов.
Ключевые слова: автоматизированное проектирование, потоки работ, программное управление, псевдо-кодовое программирование.

Библиотека БГУИР

Введение

Успешность создания любой АС в существенной мере зависит от того, «какие средства используют проектировщики для управляемого овладения её сложностью?». В оценках степени сложности систем, чаще всего, применяют различные интерпретации колмогоровской меры сложности, «как минимальной длины программы P построения системы S из её заданного исходного описания D» [Li et al, 2008]. Различия в интерпретациях обусловлены, в первую очередь, особенностями системы S, а также тем, «какое содержание вкладывается в конструкты P и D, и как это содержание специфицируется?»

В разработках АС конструкты типа P приходится создавать по ходу её проектирования, используя определённый «метод программирования M», включающий теоретическую, методическую и практическую составляющие. К одной из важнейших особенностей АС относится то, что в их разработках сложностью P необходимо овладевать не в меньшей мере, чем сложностью АС, причём исходя из того же исходного описания D. Такое отношение между АС и P позволяет вводить структуризацию в жизненный цикл создания АС, разделяя P на связную совокупность частей $\{P^i\}$, каждая из которых «отвечает» за формирование определённого состояния АС в её жизненном цикле.

Разделение P на части и, соответственно, выделение состояний $\{AC^i\}$, каждое из которых AC^i выполняет функции описания D для построения очередного состояния AC^{i+1} является традиционным приёмом овладения сложностью в разработках АС. Такое овладение находит своё материальное выражение в технологиях разработки, предоставляющих создателям АС определённые возможности построения P, а значит и частей $\{P^i\}$этого конструкта. Профессиональная зрелость конкретной технологии T*, в первую очередь, определяется тем, какие средства она предоставляет коллективу разработчиков АС для построения конструктов $\{P^i\}$, которые они же должны и оперативно исполнять.

В большинстве технологий разработок АС для достижения отмеченных целей разработчикам предоставляются средства традиционного планирования, которые не ориентированы на программные представления конструктов типа P.

В ряде технологий, например в Rational Unified Process [Кролл, 2004], традиционные средства планирования комбинируют со средствами моделирования бизнес-процессов в форме потоков работ, в том числе и представляющих их с помощью

исполняемых специализированных языков (BPEL, YAWL, XLANG, ...), но с ограниченными возможностями программного управления.

По глубокому убеждению авторов, для конструктивного учёта сложности и её оперативной минимизации (упрощения) разработчики АС должны программировать свою деятельность, применяя для этого средства унифицированного программного управления любыми потоками работ, которые порождаются в их деятельности.

Сложность является характеристикой взаимодействия проектировщиков с состояниями проекта в процессах решения задач. Проблемы со сложностью в тех взаимодействиях, которые являются источниками опасных и дорогостоящих ошибок. В этом плане наиболее опасны задачи концептуального проектирования АС.

Именно с программным управлением работой проектировщиков на концептуальном этапе связаны интересы статьи, в которой авторы представляют подход к управлению работами, в основу которого положено псевдо-кодовое программирование, ориентированное на его оперативное применение в процессах согласованного решения задач. Подход доведён до его практического применения в инструментально-моделирующей среде WIQA (Working In Questions and Answers) [Соснин, 2012].

Потоки работ концептуального проектирования

Потоки работ используются как базовые модельные представления динамики бизнес-процессов практически во всех современных технологиях разработки автоматизированных систем.

Каждый поток работ должен адекватно представлять совокупность задач проектирования, процесс согласованного решения которых приводит к осуществлению соответствующего БП. Более того, в общем случае, в процесс решения может быть вовлечена группа проектировщиков $\{D_v\}$, каждый из которых несёт ответственность за решение назначенных ему проектных задач $\{Z^S_{iv}\}$ в конкретных условиях. Обобщённое представление потока работ, раскрывающее его структурное содержание, приведено на рисунке 1.

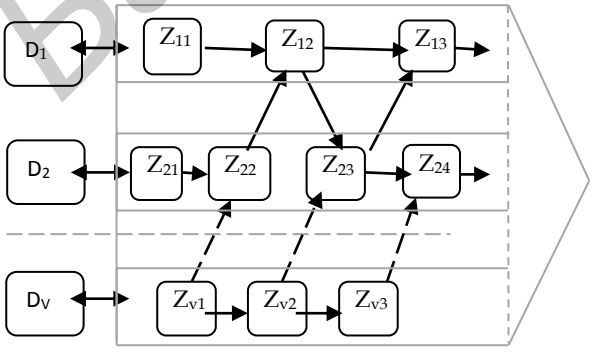

Рисунок 1 - Пример структуры потока работ

Основное предназначение потоков работ, как моделей, − автоматизация бизнес-процессов, в которой выделяются автоматизация потоков работ, автоматизация согласованного исполнения задач в потоках и автоматизация решения задач по образцам, причём во всех видах автоматизации компьютеризуется работа лиц, вовлеченных в исполнение бизнес-процессов [Held et al., 2009].

В теории и практике потоков работ, которые не привязаны к конкретным видам деятельности, среди выделенных направлений автоматизации основное внимание уделяется первым двум из-за намеренного абстрагирования от того, в какие действия человека и с чем выливается решение назначенных ему задач.

В любых его приложениях проектирование АС является деятельностью команды проектировщиков $K(\{P_v\})$, которая, автоматизируя определённую систему бизнес-процессов $S(\{БП_m\})$, должна согласованно решать определённую совокупность нормативных задач $\{Z^T_j\}$ используемой технологии T^*. По сути дела, в такой деятельности систему решений задач $S(\{Z^S_{im}\},t)$, вложенных в $S(\{БП_m\})$, пытаются представить через их «разложение» в базисе задач технологии T^*.

В любой технологии T^* её задачи объединяют в потоки работ $\{W_n(\{Z^T_{nj}\})\}$, а потоки работ объединяют в подсистемы $C^q(\{W_j(\{Z^T_{nj}\})\})$ в соответствии со специализацией этапов проектирования. Обычно задачи представлены инструкциями, раскрывающими схемы действий, каждая из которых должна быть адаптирована к соответствующим бизнес-процессам АС.

Необходимость адаптации задач $\{Z^S_{mi}\}$, обусловленная инвариантностью задач технологии к потенциальным предметным областям АС, выводит процесс проектирования на специфический класс творческих задач адаптации $\{Z^A_k\}$, включающих задачи когнитивного анализа, оценивания, принятия решений и другие задачи, требующие от проектировщика ситуативной (незапланированной заранее) интеллектуальной активности [Sosnin, 2011]. Более того, такая адаптация должна осуществляться проектировщиками согласованно. Именно необходимость согласованной интеллектуальной адаптации задач в реализации потоков работ технологии определяет важнейшую их специфику.

Реальная практика такой адаптации является источником многочисленных и разнородных ошибок и «дефектов», оказывающих негативное воздействие на процесс проектирования и его результаты. Основным источником негативов является человеческий фактор, а вернее, исполнение проектировщиками их функций в условиях недостаточной степени автоматизации активности в решении задач адаптации. Такое положение дел указывает на несовершенство существующих технологий $\{T^n\}$, подтверждаемое мировой статистикой – степень успешности разработок АС около 35 %.

Библиотека БГУИР

За последние 20 лет использовались различные средства совершенствования технологий класса $\{T^n\}$, например, модели совершенствования профессиональной деятельности [Roglinger et al., 2012] , базы опыта [Basili et al., 2001] и фабрики опыта [Henninger, 2003], унифицированный язык моделирования [Кролл, 2004], нормативные средства описания и реализации архитектурных решений [IEEE, 2000]. Шаг за шагом появлялись новые полезные средства и совершенствовались уже освоенные, однако степень успешности разработок АС почти не повышалась. А значит, проблема успешности проектирования АС всё ещё существует и поиски путей её решения за счёт технологического совершенствования деятельности проектировщиков актуальны.

Представление потоков работ в среде WIQA

Инструментальная среда WIQA изначально разрабатывалась для моделирования совокупностей технологических (нормативных) и предметных проектных задач, которые приходится решать разработчикам АС. Как источник нормативных задач были использованы типовые задачи потоков работ концептуального проектирования в технологии Rational Unified Process (RUP) [Кролл, 2004]. По его предназначению и содержанию, комплекс WIQA разрабатывался как специализированная $АС^T$ технологического типа.

Одной из важнейших особенностей концептуального проектирования в среде WIQA является потенциальное применение всех средств этого инструментария к любой задаче Z_i дерева $T(\{Z_i\})$ проектных задач $\{Z_i\}$, если в этом будет необходимость. В таком применении задача Z_i представляется её вопросно-ответной моделью (QA-моделью, $QA(Z_i)$), структурирующей процесс решения задачи в формах вопросно-ответных рассуждений. Интерфейсная форма доступа к дереву задач (Tree of Tasks, TT) и QA-моделям представлена обобщённо на рисунке 2.

Информационный потенциал дерева задач $TT(\{Z_i\})$ и связанного с ним множества моделей $\{QA(Z_i)\}$ достаточен для представления текущего состояния совокупности потоков работ $\{W_m\}$ любого проекта $П(TT(\{Z_i\}),\{QA(Z_i)\}, K(\{D_v\}), t)$, разрабатываемого коллективом $K(\{D_v\})$ проектировщиков $\{D_v\}$. Для представления и использования конструкта $K(\{D_v\})$ в комплекс WIQA встроено расширение (плагин), названный «Оргструктура».

На основе этого потенциала авторами разработан комплекс средств для представления проектов, разрабатываемых в инструментальной среде WIQA, в виде динамической схемы $S(\{W_m\}, t)$ потоков работ, предназначенной для управления процессами проектирования. В схеме $S(\{W_m\}, t)$ с каждым потоком W_m связана определённая задача Z^W_m дерева задач, которой приписан тип «поток работ» и символьное обозначение типа W.

Таким образом, в дерево задач может быть введена разметка, регистрирующая «какие задачи включены в каждый выделенный поток работ в текущем состоянии проекта?». Введение разметки приводит к новой версии систематизации $П(G(\{Z_i\}, \{Z^W_m\}), \{QA(Z_i)\}, \{QA(Z^W_m)\}, K(\{D_v\}), t)$ представления задач проектирования. Эта версия систематизации используется как основной источник информации для решения задач управления потоками работ.

Для управления потоками разработан и включён в состав комплекса WIQA ряд плагинов, обеспечивающих:

- псевдо-кодовое программирование задач, ориентированное на исполнение программ проектировщиком, выполняющим роль интеллектуального процессора (I-процессора) и использующим компьютерный процессор (K-процессор) как подчинённый процессор;
- управление прерываниями человеко-компьютерной деятельности, в частности управление прерываниями при исполнении псевдо-кодовых программ для технологических задач, кодирующих нормативные инструкции;

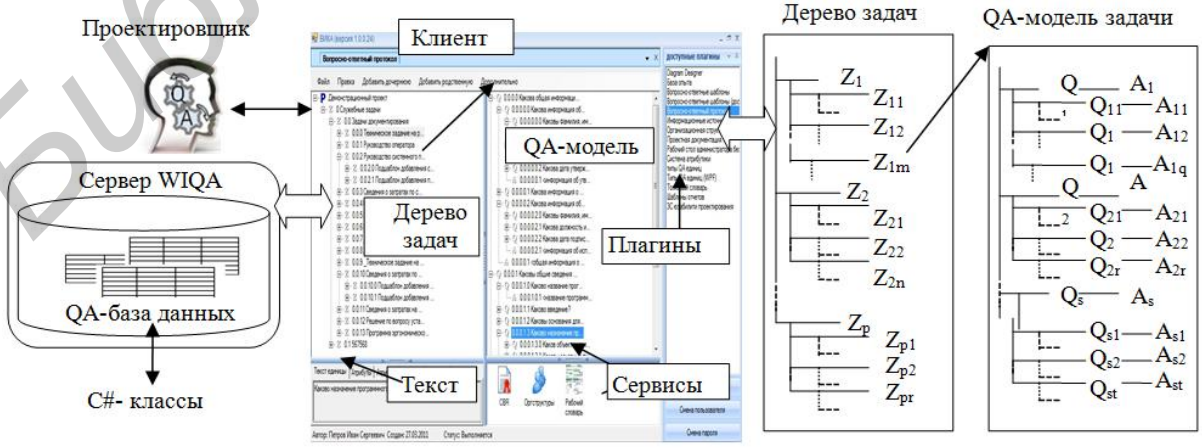

Рисунок 2 - Инструментально-моделирующая среда WIQA

Библиотека БГУИР

- оперативное использование паттернов потоков работ, псевдо-кодовые аналоги которых хранятся в специальной библиотеке.

Кроме того, для представления событий всех типов используется атрибутика объектов, входящих в дерево задач и QA-модели, в частности их уникальные индексные имена, имена проектировщиков, время последней модификации и состояния («в работе», «прервана» и «завершена»). Для учёта временных характеристик работ (решений задач всех типов) и их соответствия договорённостям разработана специализированная система «Контроля поручений». Общая картина управления потоками обобщённо представлена на рисунке 3.

Инструментарий обслуживает управляемое оперативное формирование, регистрацию, визуализацию и использование:

1. Текущего состояния дерева задач $T(\{Z_i\}, t)$, включающего:

- совокупность задач $Z^S = \{Z_{mi}\}$ предметной области АС, которые в проектируемой АС будут решать её пользователи;
- совокупность нормативных задач $Z^T = \{Z_{ni}\}$ технологии, в рамках которой коллектив проектировщиков $K(\{D_v\})$ создаёт АС;
- совокупность задач адаптации $Z^A=\{Z^A_q\}$, решаемых проектировщиками для настройки задач типа Z^T, инвариантных к проблемной области АС, в их использовании при решении задач типа Z^S;
- совокупность задач управления $Z^W=\{W^m(\{Z_{mi}\})\}\cup\{W^n(\{Z_{ni}\})\}$ и $Z^C=\{Z^W_r\}$), решение которых обслуживает работу с задачами в потоках работ, а также согласованное управление в группах потоков работ;
- объединения задач, зарегистрированных в дереве задач, в потоки работ или группу потоков, каждому из которых соответствует задача типа Z^W или Z^C, демонстрирующая динамические отношения между решениями задач во времени (средства формирования потоков работ) [Sosnin, 2012b].

2. Распределение задач, зарегистрированных в дереве, между их исполнителями $\{D_v\}$ в коллективе $K(\{D_v\})$ с использованием процедуры назначения задач (средства работ с оргструктурой коллектива и деревом задач) [Соснин, 2012].

3. Предварительное планирование сроков исполнения работ в процессах решения задач всех типов (с использованием средств контроля поручений).

4. Оперативное псевдо-кодовое программирование (с использованием средств вопросно-ответного программирования, QA-программирования) задач [Sosnin, 2012a], выбранных для решения на текущем шаге работ:

- построение псевдо-кода для назначенной проектировщику D_v задачи Z_i с возможностью регистрации псевдо-кода в виде QA-модели задачи Z_i по месту её «расположения» в дереве задач или сохранение псевдо-кода в библиотеке QA-программ (средства редактирования и сопровождения библиотек);
- использование в построении QA-программ паттернов потоков работ (средства библиотеки паттернов);
- оперативное исполнение QA-программы проектировщиком в режиме интерпретации (средства интерпретации, интерпретатор) [Sosnin, 2012c];
- компиляция QA-программы и её автоматическое исполнение компьютерным процессором (средства формирования очередей задач для их исполнения в потоках работ).

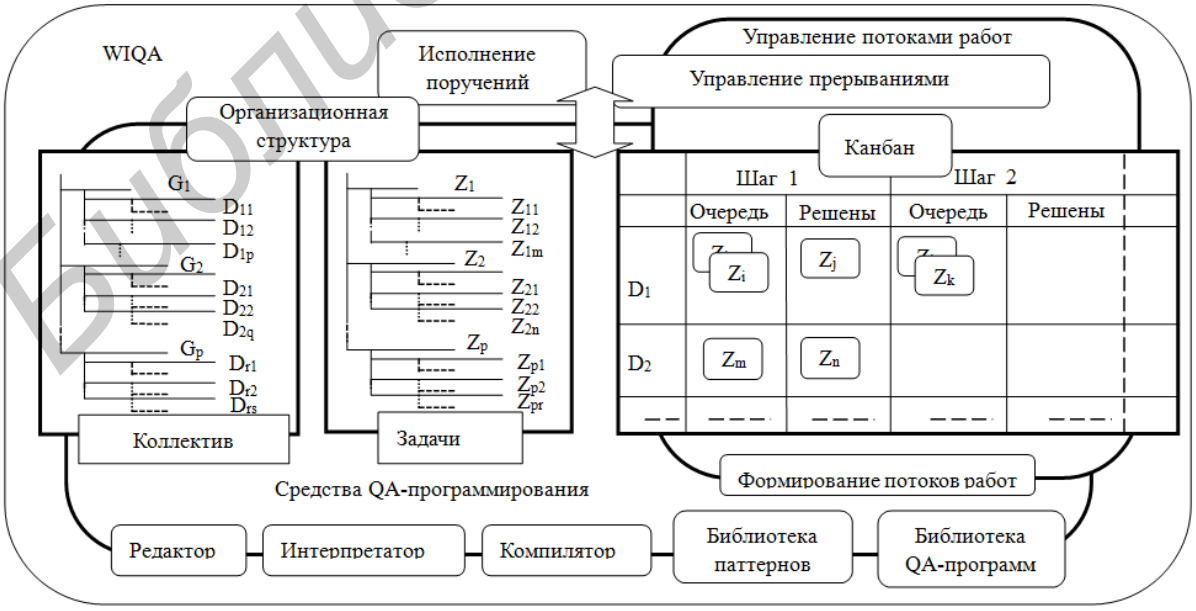

Рисунок 3 - Инструментальное обеспечение управления потоками

Библиотека БГУИР

5. Параллельное решение задач в коллективе проектировщиков на очередном шаге работ (средства работы с очередями задач с использованием визуализатора «Канбан»).

6. Псевдопараллельная работа каждого проектировщика с назначенными ему задачами (средства прерывания задач).

Для ряда перечисленных функциональных возможностей использованы ссылки на публикации, в которых раскрыты детали их реализации в среде WIQA. Из тех, для которых ссылок нет, ниже раскроем детали языка L^{WIQA} и P-модели для паттернов потоков работ и взаимодействия проектировщиков с очередями задач.

В документации комплекса WIQA используется его интерпретация как процессора, обеспечивающего регистрацию и использование моделей вопросно-ответных рассуждений (QA-рассуждений) проектировщиками в процессах решения назначенных им задач. В такой интерпретации процессора WIQA модели типов «задачи», «вопросы» и «ответы» определяются как интерактивные объекты, к которым применимы соответствующие команды.

Модели Z-, Q- и A-типов размещаются в QA-базе данных процессора, представляющей собой память (QA-память) с богатой атрибутикой «ячеек». В «ячейках» памяти можно хранить любые данные, которые будут намеренно или нет наследовать атрибутику «ячеек». Отметим, что совокупность атрибутов «ячейки» включает кроме атрибута «символьное описание», например, атрибуты «уникальное имя», «проектировщик, использующий ячейку», «тип ячейки», «момент времени записи». Более того, в комплекс WIQA встроены механизмы, позволяющие любой ячейке оперативно приписывать любые полезные атрибуты и любое их количество.

QA-память оказалась очень полезной для погружения в её структуры любых задач и их QA-моделей, в том числе и в форме QA-программ с их данными (QA-данными) и операторами (QA-операторами). Отметим, что квалификатор «QA» указывает на специфичность конструктов, подчёркивая их отличие от конструктов, родственных по предназначению. Описанная интерпретация в обобщённой форме приведена на рисунке 4.

Атрибутика, наследуемая данными и операторами, привела к созданию псевдо-кодового языка, ориентированного на исполнение QA-программ проектировщиками, которые сами же могут их строить, если в этом появилась необходимость.

В такой работе проектировщики исполняют роль, названную I-процессором [Соснин, 2012], обеспеченную полезными инструментами по образцу других ролей [Borges et al., 2012], исполняемых ими в используемой технологии.

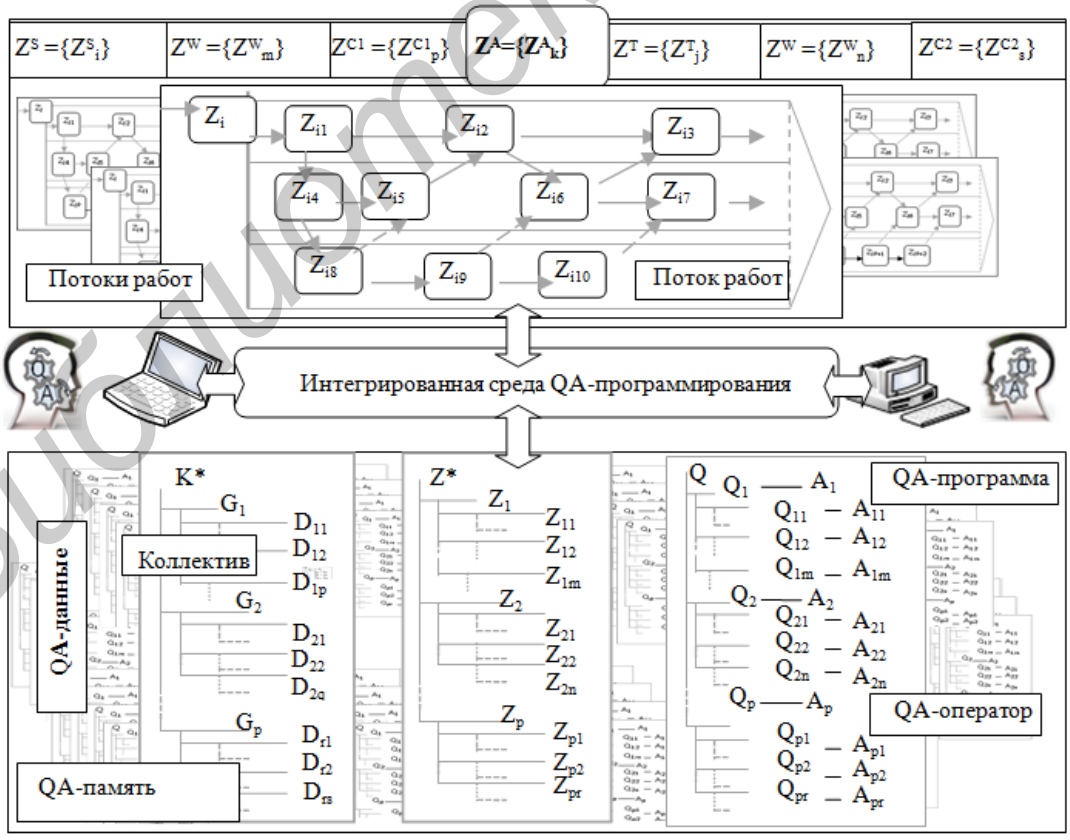

Рисунок 4 - Представление задач в QA-памяти процессора WIQA

Библиотека БГУИР

Язык псевдо-кодового программирования WIQA

Псевдо-кодовый язык QA-программирования принципиальным образом зависит от среды его употребления, из-за чего он получил имя «WIQA» и обозначение $\mathbf{L^{WIQA}}$ [Соснин, 2012]. Специфику языка $\mathbf{L^{WIQA}}$ определяют следующие его особенности:

1. Язык $\mathbf{L^{WIQA}}$ специально приближен к **естественному языку в его алгоритмическом употреблении**, обслуживающему взаимодействие человека с доступным опытом в деятельности. По этой причине язык ориентирован на решение задач, за которыми стоят прецеденты и их связные совокупности.

2. Текст исходного псевдо-кода программы на языке $\mathbf{L^{WIQA}}$ построен из представлений его структурных единиц (предложение за предложением, оператор за оператором) с помощью **QA**-данных. Так что исходный код представляет собой версию **QA**-модели задачи, для которой псевдокод был разработан. По этой причине, весь потенциал инструментария **WIQA**, применимый для **QA**-моделирования [Соснин, 2012], применим и к исходным псевдо-кодам программируемых задач.

3. Язык $\mathbf{L^{WIQA}}$ – объектно-ориентирован, поддерживает программирование задач с базами данных и программирование задач управления потоками работ в коллективе.

4. Если работа, запрограммированная на языке $\mathbf{L^{WIQA}}$, может быть «передана» компьютеру, то соответствующая QA-программа компилируется в код, исполняемый компьютерным процессором (К-процессором).

Язык разрабатывался как открытый для включения в его структуру дополнительных типов QA-данных и QA-операторов. Существующая структура языка приведена на рисунке 5.

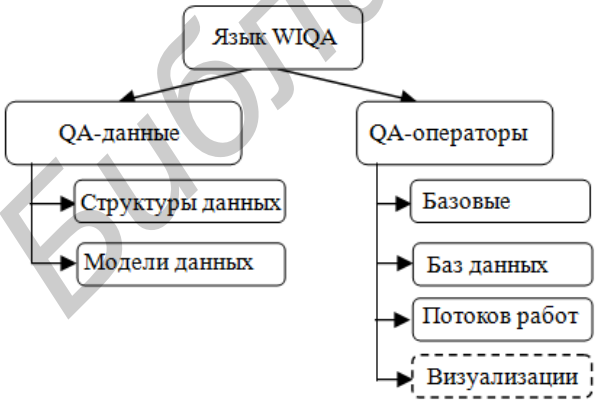

Рисунок 5 - Состав языка WIQA

На рисунке 5 отмечены отдельными группами:

- «Структуры данных», за которыми стоит эмуляция любых традиционных типов данных (скаляры, массивы, записи, множества, стеки, очереди, ...) с помощью средств QA-данных;
- «Модели данных», используемые для спецификации табличных структур и их совокупностей, а также иерархических структур;
- «Базовые операторы» традиционных псевдо-кодовых языков программирования (Appoint, Input, Output, If, GOTO другие);
- «Операторы баз данных», эмулирующие основные SQL-операторы;
- «Операторы потоков работ», эмулирующие основные операторы языков имитационного моделирования:
- «Операторы визуализации», которые разрабатываются в настоящее время для просмотра по запросам проектировщиков ячеек QA-памяти по сценариям, запрограммированным предварительно или оперативно.

Отметим, что средства объявления данных, базовый набор операторов и операторы баз данных детально представлены в публикации [Соснин, 2012]. Там же детально раскрыты спецификации роли «I-процессор» в его сопоставлении с традиционным набором ролей (около 40 ролей), используемых в технологиях разработки АС. По этой причине, ниже раскрываются детали синтаксиса и употреблений языка $\mathbf{L^{WIQA}}$ только в задачах управления потоками работ.

Псевдо-кодовое программирование паттернов потоков работ

В управлении бизнес-процессами накоплен богатейший опыт, очень важная часть которого аккумулирована в паттернах потоков работ [Aalst et al., 2005]. Роль паттернов настолько значительна, что существует международная ассоциация учёных и практиков, активность которых регистрируется на специальном Интернет-сайте (http://www.workflow patterns.com). На текущий момент на этом сайте зарегистрировано 43 паттерна, содержание которых было использовано авторами для выявления совокупности псевдо-кодовых операторов, включение которых в язык L^{WIQA} позволит использовать этот язык и в QA-программировании задач типов Z^W и Z^C.

При отборе операторов, расширяющих язык L^{WIQA} для программирования задач управления, проводились параллели между расширением и языками имитационного моделирования. Более того, для родственных операторов были сохранены их имена, устойчиво используемые в имитационном моделировании.

Для отобранной совокупности операторов произведена доработка интерпретатора и компилятора псевдо-кодов, позволяющая исполнять программы, созданные с использованием базовых операторов и операторов расширения, что позволило создать библиотек паттернов. Для примера рассмотрим паттерн «параллельное расщепление» (parallel split) [Aalst et al., 2005], одна

Библиотека БГУИР

из версий которого приведена на рисунке 6, где названы имена задач, используемые в псевдо-кодовой программе (группа – субъект, которому назначена задача).

В этой версии паттерна, исполняемого тремя (группами) проектировщиков, после того как решена Задача_1, поток расщепляется на три ветки (Задача_2, Задача_3 и Задача_4). Унификацию версий (количество расщеплений, исполнители задач) обеспечивает оператор SET, позволяющий установить значения элементам списка переменных

SET <Переменная_1>, <Значение_1>;
<Переменная_2>, <Значение_2>;...;
<Переменная_N>, <Значение_N>.

Функции переменных в данном операторе могут выполнять, как псевдо-кодовые переменные простых типов, так и элементы массивов и поля объектов. Оператор SET можно вызывать в любом месте программы для любого набора переменных.

Рассматриваемый паттерн представлен в библиотеке следующим псевдокодом:

```
//Инициализация
SET &in&, 1; &outs[0]&, 2; &outs[1]&, 3; &outs[2]&, 4; &outgroup[0]&, 1; &outgroup[1]&, 2; &outgroup[2]&, 3; &cnt&, 0
LABEL &L1&
SEIZE &outs[&cnt&]&, &outgroup[&cnt&]&
INC &cnt&
TEST L, CNT, outs.length &L1&
SET &cnt&, 0;
//Выполнение задач
LABEL &L2&
QUEUE &outs[&cnt&]&, &in&.state == done
INC &cnt&
TEST L, &cnt&, outs.length &L2&
FINISH
```

В псевдокоде паттерна присутствуют две части, разделенные комментариями. Первая часть обеспечивает распределение задач между исполнителями, в котором основная нагрузка падает на оператор SEIZE (ключевое слово заимствовано из имитационного моделирования). Этот оператор «называет» типовую работу, представленную библиотечной QA-программой со следующим псевдокодом:

```
// QA-программа SEIZE
//Установка ассоциации между группой и задачей в оргструктуре
CALL &AssignTask&
//Добавление ассоциации в систему контроля поручений
&ProjectId& := QA_GetProjectId("Workflow Data bases")
&BaseId& := QA_GetQAId(&ProjectId&, "Commissions")
//Получение идентификатора БД
&Database& := OpenDB(&BaseId&)
//Вдруг задача уже назначена? Проверка
&Query& := "SELECT GroupId FROM Commissions WHERE TaskId="
&Query& := concat (&Query&, <Задача>)
&Query& := concat (&Query&, ";") //Построение запроса
&Res& := ExecuteSQL (&Database&, &Query&)
if (&Res& <> "NULL") then return //Задача уже назначена
//Формирование запроса, добавляющего задачу в очередь
&Query& := "INSERT INTO Commission (GroupId, Task) VALUES ("
&Query& := concat(&Query&, <Исполнитель>)
&Query& := concat(&Query&, ", ")
&Query& := concat(&Query&, <Задача>)
&Query& := concat(&Query&, ");")
//Выполнение запроса – ассоциирование задачи
ExecuteSQL (&Database&, &Query&)
```

Принципиальной составляющей псевдокода SEIZE является назначение задачи исполнителю с использованием плагина «Оргструктура» и дерева задач текущего проекта:

1. *procedure &AssignTask& . вызывается для назначений по каждой задаче (_Z1, ...)*
2. *Запустить плагин «Оргструктура»*
3. *Выбрать группу <Исполнитель>, которой требуется назначить задачу*
4. *В правой части окна нажать кнопку Выбор проекта*
5. *Выбрать проект, содержащий требуемую задачу*
6. *Найти задачу <Задача> в дереве проекта*
7. *Нажать кнопку «Назначить задачу»*
8. *endproc &AssignTask&*

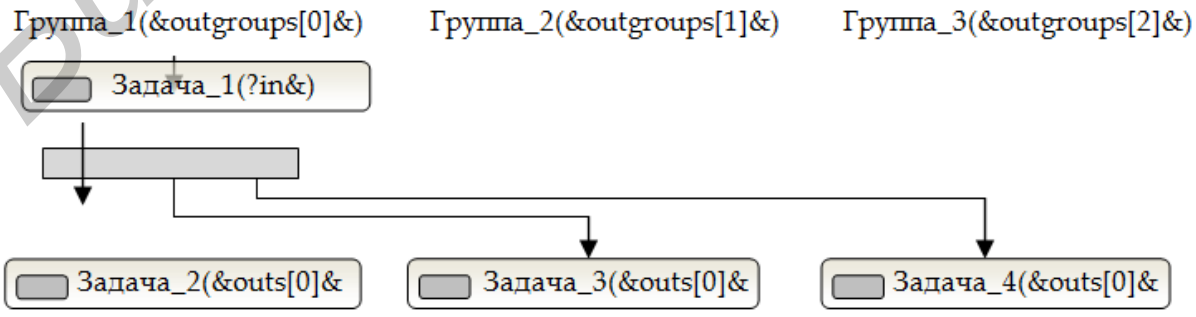

Рисунок 6 - Структура паттерна «Parallel split» (Параллельное расщепление)

Библиотека БГУИР

Каждое произведённое назначение регистрируется в основной таблице плагина «Контроль поручений», атрибутика которой включает «Уникальный идентификатор поручения», <Идентификатор исполнителя поручения>, <Тип поручения>, < Идентификатор назначенной задачи>, <Резолюция >, < Важность>, <Срок исполнения >, <Контрольные точки > и <Время выполнения задачи».

Библиотека паттернов разработана для облегчения оперативного программирования потоков работ, необходимость в которых выявлена в процессе проектирования. Одним из важнейших источников таких конструктов является пошаговая детализация, по ходу которой, например, задача Z_J определённого уровня (пусть уровня J), разбивается на связную совокупность задач $C(\{Z_{J+1}\})$ уровня J+1. Разумеется, до QA-программирования такого потока динамические отношения между подчинёнными задачами должны быть представлены схемой потока W^k.

Управление очередями задач

Как уже отмечалось выше, QA-программирование управляющей программы начинается с подготовки потока W^x к исполнению, включающей назначения задачам исполнителей и установлению плановых характеристик для их решения.

Предположим, что эта работа и все необходимые остальные работы учтены, и QA-программа для потока W^k построена. Разумеется, построенную программу необходимо проверить. Одной из версий проверки является представление QA-программы в базисе паттернов проектирования, псевдокоды которых заимствуются из библиотеки паттернов и встраиваются в проверяемую программу.

Именно такая нагрузка и возлагается в совокупности средств программного управления на библиотеку паттернов потоков работ. Сам факт разложения QA-программы потока в базисе паттернов является важным аргументом в пользу корректности построенной QA-программы.

Предположим, что QA-программа для потока (пусть W^k) построена и проверена, причём не только через её разложение в базисе паттернов. После этого её следует исполнить. В авторской версии программного управления потоками исполнение завершающей фазы QA-программы потока осуществляется компилятором, который по информации о задачах, вложенной в псевдокод потока, дополняет соответствующие очереди задач новыми составляющими. Другими словами, программное управление потоками работ нацелено на формирование очередей задач для каждого члена коллектива разработчиков.

Более того, авторы предлагают различать и строить очереди задач в виде специализированных программ двух типов (M^1-программ и M^2-программ), состоящих из операторов, каждый из которых может исполняться только в случае истинности зафиксированного в операторе условия.

То есть каждый элемент в любой очереди независимо от её типа оформляется и интерпретируется как условный оператор

If <условие доступа к элементу очереди> Then <Доступ к задаче по её идентификатору>,

исполняемый проектировщиком в роли I-процессора (рисунок 7).

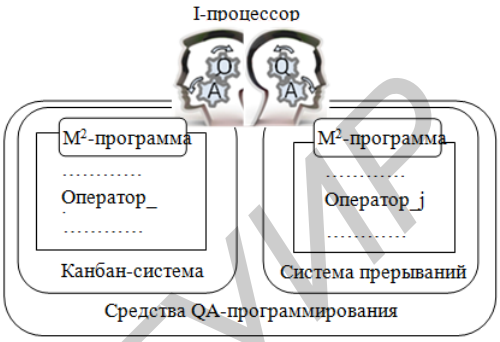

Рисунок 7 - Взаимодействие I-процессора с очередями задач

Различия между очередями M^1- и M^2-типов состоит в том, что:

- в M^1-программах условия доступа регистрируют логику отношений между задачами потока во времени, управляя тем самым параллельно-согласованным решением задач группой проектировщиков;
- в то время как условия операторов M^2-программ управляют псевдопараллельным решением задач конкретным проектировщиком, каждая из которых может быть прервана по ходу решения, если в этом возникла необходимость.

Использование очередей двух типов позволяет отделить:

- визуализацию общей картины исполнения задач группой проектировщиков;
- от интерактивных взаимодействий проектировщика с задачами, которые он может выполнять псевдопараллельно.

Для формирования очередей в расширение языка L^{WIQA} введён оператор QUEUE. Основное назначение оператора зарегистрировать условие, истинность которого открывает возможность для начала работы с задачей

QUEUE <Задача>, <Условие>.

За этим оператором также как и за оператором SEIZE стоит библиотечная QA-программа, псевдо-код которой выглядит следующим образом:

&ProjectId& := QA_GetProjectId("WorkflowDatabases")
&BaseId& := QA_GetQAId(&ProjectId&, "Commissions")
&Database& := OpenDB(&BaseId&) //Получение идентификатора БД
&Query& := "SELECT GroupId FROM Commissions WHERE TaskId="

Библиотека БГУИР

```
&Query& := concat (&Query&, <Задача>)
&Query& := concat (&Query&, ";") //Построение запроса
&Res& := ExecuteSQL(&Database&, &Query&) //Выполнение запроса
//Заполнение переменных, используемых функциями управления очередями
&QueueId& := UnpackInt(&Res&)
&TaskId& := <Задача>
//Автоопределение приоритета для размещения задачи в конце очереди
&Priority& := -1
&TaskCondition& := "<Условие>";
//Вызов функции добавления задачи &TaskId& в очередь &QueueId&
CALL &AddTask&
```

Процедура &AddTask& подсистемы управления очередями задач выполняет взаимодействие с таблицей базы данных очередей задач, выполняет проверки, и после них заносит в таблицу очередей новую задачу.

Визуализация общей картины осуществляется с помощью средств интерактивной «доски» Канбан [Wang et al., 2010], фиксирующей для выбранной совокупности шагов проекта состояния работ с задачами «в очереди работ» или «завешена». Идентификаторы задач регистрируются в очередях системы Канбан в результате выполнения соответствующих операторов управления QUEUE.

Псевдопараллельное решение задач обслуживается средствами системы прерываний I-процессора с учетом информации по контролю за поручениями, которая используется для оперативных вычислений динамических приоритетов. Коррекция приоритетов осуществляется автоматически при очередном обращении проектировщика к очереди. Приоритеты помогают в выборе, но не диктуют его проектировщику, который свободен в действиях такого типа.

Заключение

Разработанные и представленные в статье средства управления потоками работ предназначены для программного формирования очередей задач, взаимодействие с которыми обеспечивает согласованную работу коллектива проектировщиков в разработке АС. Для программирования задач управления разработано расширение псевдо-кодового языка L^{WIQA} до его версии, позволяющей создавать программные модели, как для процессов решения задач, так и для управления их согласованным решением в потоках работ. Операторы расширения (по своему содержанию, а не по реализации) частично заимствованы из традиционных языков имитационного моделирования.

В разработке средств учитывалась конструктивная работа проектировщиков со сложностью АС с позиций колмогоровской меры, ориентированной на «программное построение сложных систем». Использование в таких построениях псевдо-языковых средств способствует пониманию планируемой и исполняемой работы, а также её потенциальному упрощению за счёт создания прототипов человеко-компьютерной деятельности.

Заключение

Представленные в статье средства управления потоками работ обеспечивают единообразный подхода к автоматизации согласованного управления потоками работ, задачами в потоках и действиями проектировщиков в решении нормативных задач на базе псевдо-кодового программирования в инструментальной среде WIQA.

Для программирования задач управления разработано расширение псевдо-кодового языка L^{WIQA}, позволяющее создавать программные модели, как для процессов решения задач, так и для управления их согласованным решением в потоках работ. Операторы расширения, по своему содержанию, а не по реализации, частично заимствованы из традиционных языков имитационного моделирования.

Разработанные средства управления обеспечивают конструктивную работу проектировщиков со сложностью АС с позиций колмогоровской меры, ориентированной на программируемое построение сложных систем. Использование в работе псевдо-языковых средств способствует пониманию планируемых и исполняемых работ, а также их потенциальному упрощению за счёт создания прототипов и моделей прецедентов.

Библиографический список

[Кролл, 2004] Кролл П. Rational Unified Process – это легко: Руководство по RUP для практиков / П. Кролл, Ф. Крачтен– М.: КУДИЦ-Образ, 2004. – 427 с

[Соснин, 2012] Соснин П.И. Программирование человеко-компьютерной деятельности. /П.И.Соснин. – Saarbruken: Lambert Academic Publishing, 2012 – 343 с.

[Aalst et al., 2005] Aalst M., Hofstede A.H.M. and M. Dumas. Patterns of Process Modeling./ In Process-Aware Information Systems: Bridging People and Software through Process Technology, Wiley & Sons, (2005), pp 179-203.

[Basili et al., 2001] Basili A. V., M. Lindvall M. and Costa P. Implementing the experience factory concepts as a set of experience bases, In Proc. of the 13 th International Conference on Software Engineering & Knowledge Engineering, (2001), 102-109.

[Borges et al., 2012] Borges P., Machado R.J. & Ribeiro P. Mapping RUP Roles to Small Software Development Teams, In Proc. of International Conference on Software and System Process (ICSSP), Portugal, (2012), pp. 190-199.

[Charette, 2005] Charette R.N. Why software falls, IEEE Spectrum, 42(9), (2005), 36-43.

[IEEE, 2000] IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. Institute of Electrical and Electronics Engineers, Sept. 2000. IEEE Std 1471-2000

[Li et al., 2008] Li M. and P. M.B.Vitanui, An Introduction to Kolmogorov Complexity and Its Applications. Series: Text in Computer Science, 3rd ed., Springer, 2008.

[Held et al., 2009] Held M. and Blochinger W. Structured collaborative workflow design, Future Generation Computer Systems, 25(6), (2009), 638-653.

Библиотека БГУИР

[Henninger, 2003] Henninger S. Tool Support for Experience-based Software Development Methodologies, Advances in Computers, 59, (2003), 29-82.

[Roglinger et al., 2012] Roglinger M., Poppelbuth J. & Becker J., Maturity models in business process management, Business Process Management Journal, Vol. 18 (2), (2012), pp. 328 – 346.

[Sosnin, 2011] Sosnin P. Question-Answer Shell for Personal Expert System // Chapter in the book "Expert Systems for Human, Materials and Automation." Published by Intech, (2011), pp. 51-74.

[Sosnin, 2012a] Sosnin P. Question-Answer Approach to Human-Computer Interaction in Collaborative Designing. Chapter in the book "Cognitively Informed Intelligent Interfaces: Systems Design and Development" Published IGI Global, (2012), pp. 157-176.

[Sosnin, 2012b] Sosnin P. Pseudo-code Programming of Designer Activity in Development of Software Intensive Systems// In Proc. of the 25-th International conference on Industrial Engineering and other Applications of Applied Intelligent Systems (IEA/AIE 2012), Dalian, Chine, (2012), pp. 457-466.

[Sosnin, 2012c] Sosnin P. Experiential Human-Computer Interaction in Collaborative Designing of Software Intensive Systems// In Proc. of the 11-th International conference on Software Methodology and Technics (SoMeT'2012), Genua, Itali, (2012), pp. 180-197.

[Wang et al., 2010] Wang J. X . Kanban: Align Manufacturing Flow with Demand Pull, Chapter in the book: Lean Manufacturing Business Bottom-Line Based, CRC Press (2010), 185-204.

PROGRAM MANAGEMENT OF WORKFLOWS IN CONCEPTUAL DESIGNING OF AUTOMATED SYSTEMS

Sosnin P.I. *, Lapshov Y.A. *, Maklaev V.A.**

** Ulyanovsk State Technical University, Ulyanovsk, Russia*

sosnin@ulstu.ru

y.lapshov@ulstu.ru

*** Federal Science-Production Centre "MARS", Ulyanovsk, Russia*

mars@mv.ru

The paper presents means and the language of pseudo-code programming of workflows, intended for algorithmic representation of business-processes in conceptual designing of automated systems (AS). The suggested approach facilitates the decreasing of the complexity in processes of designing.

INTRODUCTION

Programming of workflows is the useful way for reducing the complexity of the designer work in collaborative conceptual designing of automated systems (AS). Efficiency of collective works can be essentially increased if human activities in workflows will be based on executions of a special kind of programs by designers any of which plays a role of an "intellectual processor" when it can lead to positive effects. Such role is a model of a designer behavior providing the adjustment of workflows to the content of AS. This role is additional to the traditional set of roles used in development technologies of AS. The role implementation is being supported by specialized means of pseudo-code programming embedded to the toolkit WIQA (Working In Questions and Answers) the use of which is oriented on conceptual designing of AS.

Main Part

The successful creation of any AS in an essential measure depends on what means are used by designers for operated mastering by the system complexity. In general sense the complexity (or simplicity) of AS reflects the degree of difficulty for designers in their interactions with definite models of AS (or its components) in solving the definite tasks.

This paper presents new means for mastering of the complexity which are based on pseudo-code programming of workflows in collaborative conceptual designing. Programming of the conceptual activity helps to increase the level of its automation and by that to reduce the complexity of collaborative works and quantity of semantic defects in designer solutions. The suggested means manage the designer who should fulfill the role of I-processor working under control of QA-programs in decisions of typical project tasks.

Possibilities of QA-programming are implemented in the instrumental system WIQA, supporting the work of designers with the precedent base in the corporate network. Means of QA-programming include the interpreter of QA-programs for their execution by designers and the compiler of QA-programs for the computer execution.

The suggested pseudo-code language is defined and built as the object-oriented language the potential of which is sufficient for the expression of the necessary semantics. QA-programming is being applied as to the precedents already mastered by designer and for the precedents being created during designing.

Developed means support the creation of QA-programs for the management of workflows. To provide such possibility the traditional set of pseudo-code operators are evolved for programming the dynamics relations between tasks embedded to workflows. QA-programs of this type of are compiled to M^2-programs which support the work with queues of project tasks.

Similar approach is used for the pseudo-parallel execution of appointed tasks by I-processor. The queue of interrupted tasks is interpreted as programs of M^1-type any of which combines conditional operators activated by events also.

CONCLUSION

The mastering of the complexity is the reliable way to increase the degree of the success in designing of AS. The useful means of this way is the program management of workflows in the conceptual stage of designing.

The complexity is being reduced because the library of programmed assets is the source of automated resources each of which can be included to the program of designer activity through calling the name of the asset. In WIQA-environment the assets are embedded to the repository which is a kernel of Experience Factory based on models of precedents.

Библиотека БГУИР