УДК 004(4’22+054)

Чайников С.И.[1], Солодовников А.С.[2]

[1] К.т.н., доцент, каф. системотехники,
ХНУРЭ, г. Харьков, Украина, serhii.chainikov@nure.ua

[2] К.т.н., доцент, каф. медицинской и биологической физики и медицинской информатики,
Харьковский национальный медицинский университет, г. Харьков, Украина, andrew.sldv@gmail.com

МЕТОДЫ СТРУКТУРНОГО СИНТЕЗА И АВТОМАТИЗИРОВАННОГО КОНФИГУРИРОВАНИЯ ПРОГРАММНОЙ АРХИТЕКТУРЫ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Проведен анализ методов структурного синтеза и кастомизации программного обеспечения информационной системы. Указывается, что современные методы не удовлетворяют требованиям к эффективной адаптации программного обеспечения под изменяющиеся во времени требования конечного пользователя. Показано, что при разработке формальных графовых моделей программной архитектуры с использованием существующих методов эволюционные изменения требований конечного пользователя обычно не рассматриваются, что приводит к трудностям при решении задачи кастомизации. Несоответствие между возможностями существующих технологий проектирования и практической необходимостью адаптации программного обеспечения приводит к возникновению проблемы разработки эффективных формальных подходов к кастомизации. Анализ методов подтверждает актуальность решения задачи разработки эффективных формальных подходов к кастомизации программного обеспечения с учетом специфики конкретного предприятия и конкретного рабочего места путем использования ярусно-параллельных графовых моделей программной архитектуры.

ГРАФОВАЯ МОДЕЛЬ, АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, СТРУКТУРНЫЙ СИНТЕЗ, ЯРУСНО-ПАРАЛЛЕЛЬНАЯ ФОРМА, BACKTRACKING, КОНТРОЛЬНАЯ ТОЧКА, AGILE, GRID, TDD, СЕРВИС-ОРИЕНТИРОВАННЫЙ ПОДХОД

Чайніков С.І., Солодовніков А.С. Методи структурного синтеза й автоматизованого конфігурування програмної архітектури інформаційної системи. Проведено аналіз методів структурного синтезу і кастомізації програмного забезпечення інформаційної системи. Вказується, що сучасні методи не задовольняють вимогам до ефективної адаптації програмного забезпечення під змінні в часі вимоги кінцевого користувача. Показано, що при розробці формальних графових моделей програмної архітектури з використанням існуючих методів еволюційні зміни вимог кінцевого користувача зазвичай не розглядаються, що призводить до труднощів при вирішенні задач кастомізації. Невідповідність між можливостями існуючих технологій проектування і практичною необхідністю адаптації програмного забезпечення призводить до виникнення проблеми розробки ефективних формальних підходів до кастомізації. Аналіз методів підтверджує актуальність вирішення задачі розробки ефективних формальних підходів до кастомізації програмного забезпечення з урахуванням специфіки конкретного підприємства і конкретного робочого місця шляхом використання ярусно-паралельних графових моделей програмної архітектури.

ГРАФОВА МОДЕЛЬ, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СТРУКТУРНИЙ СИНТЕЗ, ЯРУСНО-ПАРАЛЕЛЬНА ФОРМА, BACKTRACKING, КОНТРОЛЬНА ТОЧКА, AGILE, GRID, TDD, СЕРВІС-ОРІЄНТОВАНИЙ ПІДХІД

S.I. Chainikov, A.S. Solodovnikov. Methods of structural synthesis and automated configuration of the program architecture of information system. Authors represent analysis of the methods of structural synthesis and customization of the information system software. It is indicated that modern methods do not satisfy the requirements for effective adaptation of software for time-varying end-user requirements. It is shown that when developing formal graph models of software architecture using existing methods, evolutionary changes in end-user requirements are usually not considered, which leads to difficulties in solving the problem of customization. The mismatch between the capabilities of existing design technologies and the practical needs to adapt software leads to the problem of developing effective formal approaches to customization. The analysis of the methods confirms the relevance of solving the problem of developing effective formal approaches to customization of software, taking into account the specifics of a particular enterprise and a particular workplace by using tier-parallel graph models of software architecture.

GRAPH MODEL, SOFTWARE ARCHITECTURE, STRUCTURE SYNTHESIS, MULTILEVEL STRUCTURE, BACKTRACKING, CONTROL POINT, AGILE, GRID, TDD, SERVICE BASED APPROACH

Введение

Многие области человеческой деятельности в связи с тенденцией к усложнению за последнее время требуют поддержки информационных технологий (ИТ) с целью оптимизации и автоматизации труда. В рамках конкурентной структуры рынка программного обеспечения (ПО) немаловажную роль в процессах проектирования и разработки играет не только качество, надежность, информационная безопасность, но и скорость формирования готового программного продукта. Так же заказчики предъявляют высокие

требования к скорости выполнения функций самим ПО. Особенно это характерно для информационных систем (ИС), характеризующихся структурной, функциональной, информационной сложностью, сложной динамикой поведения. Проектирование и разработка подобного рода ИС требует значительных трудовых, временных затрат. В связи с увеличением сложности ИС, согласно стандарту ISO/IEC 12207-2008, усложняются процессы проектирования программной архитектуры, менеджмента конфигурации, менеджмента повторного применения программ и сопровождения ПО ИС. Поэтому на современном этапе активно развиваются автоматизированные методы синтеза программмной архитектуры, её компоновки и конфигурирования. Возникает задача кастомизации ПО, выражающаяся в адаптации программной архитектуры и функционала программного продукта к требованиям конечного пользователя.

1. Методы структурного синтеза и автоматизированного конфигурирования программной архитектуры

Один из распространенных подходов к формированию архитектуры ПО – применение сборочного подхода или использование технологии композитных приложений, которая показывает свою эффективность при использовании готовых программных компонентов сторонних разработчиков [1]. Тем не менее, в некоторых случаях существуют проблемы, связанные с принципами стыковки программных компонентов, обеспечением совместимости их функций. Решение этих проблем происходит: а) в рамках использования ограниченного числа компонентов, известных пользователю; б) в случае принадлежности компонентов одному разработчику, когда их технологическая и методологическая совместимость изначально обеспечена [2]. В этих случаях логично использование проблемной ориентации целевого ПО [3], применяемого для задач моделирования, контроля, анализа, автоматизированного управления, которое ограничено узкой предметной областью (ПрО). Синтез таких программных систем или комплексов программных средств (ПС) возможен на базе существующих программных компонент, сервисов и программных модулей.

Сборка ПО осуществляется в ручном, автоматическом или полуавтоматическом (автоматизированном) режимах [4].

Автоматический режим, хотя и позволяет снизить время разработки программы, все же обладает рядом недостатков в сравнении с автоматизированными методами, а именно – сложность генерации ПО для распределенных или параллельных вычислительных систем (ВС) и при наличии недетерминированных, трудно формализуемых ПрО.

В общем смысле выделяют два подхода к синтезу программной архитектуры ИС, основываясь на степени формализации исходной модели ПрО: 1) логический и 2) структурный синтез [4]. Логический синтез программной архитектуры базируется на математическом исчислении, представляющем закономерности функционирования объектов и их взаимосвязи, и трудно применим для нетривиальных ПрО. Структурный синтез обладает большими преимуществами в таких случаях и позволяет использовать наглядное представление структуры.

Логичным совмещением формальных и неформальных средств описания архитектуры ИС является архитектурный фреймворк [5], вмещающий в себя конвенции, принципы и методы описания архитектуры.

Для выявления особенностей и подходов к автоматизации процесса синтеза программ рассмотрим существующие методы.

Для процесса проектирования программной архитектуры ИС за основу может быть взята одна из существующих технологий проектирования (SADT, IDEF, SSADM, Meris) [6], основываясь на критерии схожести интерпретации этапов жизненного цикла (ЖЦ) ИС. Однако для обеспечения автоматизации процесса синтеза архитектуры в качестве исходной информации используется формализованное описание ПрО, формализованное представление программной архитектуры, а также требования конечного пользователя к ПО.

Такой формальный подход реализован на базе совокупности формальных документов, адекватно отражающих ПрО, в виде инструментария – генератора проектов, который позволяет на конечных этапах генерировать программный код системы и выполнять технологическую сборку [7].

Разработки в данном направлении велись с 80-х годов [20, 9]. Однако, в случае использования метода генерации проектов имееют место недостатки, связанные с употреблением генерируемых скриптов вместе с текстами программного кода для сборки программ проекта по исходным текстам в соответствии с выбранной платформой. Это является причиной разработки и поддержки дополнительного ПО, которое анализирует полученные скрипты.

В 80-х годах в качестве инструментария для генерации программной архитектуры была предложена диалоговая система, позволяющая осуществлять синтез ПО для промышленных объектов [10]. Ключевыми особенностями данной разработки является оптимизация процесса

синтеза программной архитектуры путем: 1) запоминания состояния задачи на любом этапе с последующим восстановлением; 2) проверки исходных данных задачи до ее решения, в процессе решения и после него; 3) оперативного ввода исправлений в исходные данные; 4) предоставления пользователю возможности многосеансной работы. Такой функционал позволяет прорабатывать различные стратегии решения задачи, минимизировать время ожидания решения задачи за счет снижения количества ошибок, приводящих к сбою ВП и разбивать последовательность действий пользователя на этапы (сеансы) с длительными временными перерывами между ними. Для осуществления синтеза программной архитектуры делается упор на развитие проблемно-ориентированного языка.

В 90-х годах было предложено использование диалоговых систем для автоматизированного формирования ВП на основе маршрутов, выделяемых на графовой модели вычислений [11]. Однако развитие диалоговых систем пошло в сторону проектирования диалога на базе естественного языка и в дальнейшем получило развитие в применении искусственного интеллекта и баз знаний [12].

В настоящее время также известны некоторые системы автоматизированного технологического проектирования, работающие в диалоговом режиме, например, «ТехноПро» [13]. Все же на современном этапе уделяется мало внимания вопросам оптимизации диалогового интерфейса пользователя в смысле разграничения функциональной нагрузки между пользователем и ПО ИС. В целях повышения эффективности работы пользователя применяются попытки оценивания физиологических показателей [28].

Выделяют также технологию автоматического синтеза программной архитектуры с использованием онтологии прецедентов [15]. Онтологическое описание позволяет накапливать опыт разработки, выполнять автоматическую классификацию программ на основе их спецификаций и выполнять построение программ путем адаптации известных решений [16]. Онтологическое моделирование ПрО получило развитие в области GRID-технологии, облачных вычислений, технологий e-Science [17].

Другое направление в автоматическом синтезе архитектупы ПО связано с развитием технологии генетических алгоритмов, позволяющих определять структуру управляющего автомата, который в свою очередь является системой вложенных и взаимовызываемых автоматов [18]. Такой подход реализует технологию автоматного программирования и обладает такими преимуществами как автоматизация процесса верификации, документируемости, упрощение процедуры внесения изменений. Автоматный подход находит широкое применение не только в синтезе программного кода [19], но и в управлении поведением самой системы, запущенной на выполнение.

Заслуживают внимания методы и средства автоматического построения параллельных программ с использованием технологии CUDA по непроцедурным спецификациям [20]. Такая технология базируется на методах декларативного программирования, что позволяет получать программу с высоким уровнем абстракции с отражением самого метода решения, а не его реализацию при конкретных условиях. Недостаток такого подхода – ограниченная применяемость в силу недостаточной универсальности методов, неприменимых для другого аппаратного обеспечения и ПрО.

Одним из эффективных подходов на современном этапе является сервис-ориентированный подход. Он требует применения архитектурных фреймворков и банков видов моделей, методов, знаний, правил и алгоритмов для конструирования ИС в рамках выбранной методологии. Для описания программной архитектуры ИС широко используются ADL-языки (Architectural description languages) [21]. ADL-языки используются в качестве средств описания архитектурных спецификаций, их интеграции с целевыми моделями ИС, описанных аспект-ориентированными графами [22]. В случае параметрического синтеза ПО ИС применяются модели многослойного графа [23]. Кроме специализированых ADL-языков также применяются UML нотации для описания программной архитектуры.

Применение архитектурных шаблонов и проблемно-ориентированных языков описания архитектуры находит применение в развитии технологии построения композитных приложений [24].

С усложнением ПрО увеличивается сложность аппаратного и программного обеспечения. Повышаются требования к эффективности и производительности ПО (стандарт ISO/IEC 25041:2012). Это требует применения альтернативных технологий увеличения вычислительной мощности [25]. Решение проблем оптимизации ВП сводится к технологиям организации распределенных и параллельных вычислений. Однако в этой связи растет сложность проектирования и разработки качественного ПО. В случае невозможности непосредственного использования конечным пользователем ИС с проблемной ориентацией на базе многопроцессорной или распределенной архитектуры прользователю предлагается использование GRID-технологии и предоставление вычислительных мощностей компьютерного кластера в качестве сервиса [26].

Сервисно-ориентированные технологии организации кластерных вычислений являются на данный момент наиболее перпективными. Они порождают новое ответвление – облачные технологии. Однако в области сервисно-ориентированных технологий присутствует существенная проблема – обеспечение безопасности данных, находящихся во владении сторонних организаций. Среди известных типов 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 [27].

В случае использования итерационных моделей ЖЦ ИС и динамически формируемых требований используются Agile-технологии, которые показывают свою эффективность для небольших компаний-разработчиков ПО. В случаях средних и крупных компаний, которые ведут разработку сложных программных систем с заданными высокими требованиями к надежности, точности и эффективности, применяются технологии, учитывающие функциональные и нефункциональные требования конечного пользователя. Программные спецификации, составляемые на основе требований, используются в качестве основы для разработки ПО через тестирование TTD [28] и с учетом поведенческих свойств ПО – BDD [29, 30]. Однако постепенный рост дополнительной функциональности готовых программных продуктов в процессе его технической поддержки приводит к неконтролируемому разрастанию программной архитектуры, увеличению сложности программ, что влечет за собой увеличение стоимости или прекращение сопровождения ПО. Это становится проблемой для технологий гибкой и интенсивной технологий разработки в современных условиях рынка ПО [31]. В качестве решения проблемы неконтролируемого разрастания функциональности программ предлагается использовать мониторинг актуального состояния ПО на предмет идентификации устаревшей функциональности и удалении ненужных функций из программной архитектуры. Автоматический процес сокращения функциональной сложности базируется на отслеживании изменения значимости функций приложения во времени от версии к версии в компании разработчика путем фиксации частоты использования функций ПО конечным пользователем.

Рост функциональной сложности, являющийся одной из причин повышения стоимости сопровождения программных продуктов, обусловливает необходимость применения также и другого подхода в решении этой проблемы. Кстомизация ПО для бизнес-процессов и ИТ-инфраструктуры является выходом из сложившейся ситуации, например, для больших программных комплексов такого класса как ERP-системы [32]. Поскольку такие системы требуют нескольких месяцев или лет для развертывания, внедрения и начала успешной эксплуатации [33]. Кастомизация подразумевает адаптацию ПО к организационной структуре ИС, основываясь на использовании сервис-ориентированной архитектуры и сервис-доминантной логики (Service Dominant Logic) [32]. Для автоматизации процессов кастомизации и снижения функциональной сложности ПО требуется применение формального аппарата –графовых модели программной архитектуры разрабатываемых ИС.

На современном этапе для решения проблем кастомизации и динамического конфигурирования компонентов ПО применяется развитая технология динамических линеек программных продуктов DSPL [34]. Такие системы адаптируются не только к изменяемым требованиям пользователя, но также и к среде функционирования, позволяя изменять свою функциональность без перезагрузки системы. DSPL технология применяется для разработки саморегулирующихся систем. Технология DSPL может базироваться на использовании сервис-ориентированной архитектуры программного продукта [35]. Недостаток этих технологий – недостаточное обеспечение безопасности данных.

По данным отчетов за 2015 и 2016 годы компании Panorama Consulting Solutions о применяемых технологиях в проектировании и разработке программных систем ERP класса наблюдаются тенденции значительного снижения процента использования сервис-ориентированой технологии в рамках модели «приложение как сервис» SaaS (с 33% до 17%); увеличение процента использования ERP-платформы на базе технологии ERP-облако (с 11% до 27%). При этом остается неизменным процент использования локального развертывания и использования ПО (56%) [36, 37]. Согласно данным этого же отчета предприятия, преобретающие программный продукт, отказываются от применения модели SaaS и технологии облачных вычислений по причине недостаточного уровня защиты данных: процент неудовлетворенности возрос с 20% до 29% на фоне снижения других причин (рисунок 1, а).

Для предприятий, которые внедряют ERP-системы, важным показателем является процент кастомизации приобретенного ПО, то есть процент доработок, осуществляемых при адаптации программного функционала к условиям бизнес-процессов предприятий и требованиям пользователей. В соответствии с отчетами 2015-2016 гг.

а *б*

Рис. 1. Данные отчетов: *а* — причины отказа от использования облачных технологий; *б* – требуемая кастомизация приобретенного ПО

наибольшее количество предприятий (41%) требуют 11-25% доработок [36, 37]. Однако на 2016 год их количество снизилось до 31% на фоне медленного увеличения количества предприятий (с 22% до 23%), которые требуют 26-50% кастомизации своего программного продукта (рисунок 1, б). Одновременно с этим, благодаря высокой стандартизации и унификации бизнес-процессов произошло значительное увеличение числа предприятий, требующих небольшие изменения в ПО после преобретения (уровень кастомизации 1-10%). Тем не менее, приведенные данные говорят о сохраняющейся актуальности использования своих технологических решений при проектировании и разработке ПО, которое развертывается локально на предприятиях, но, подобно DSPL или сервис-ориентированным технологиям, позволяет обеспечить высокую гибкость, динамическую конфигурируемость, адаптируемость программной архитектуры и функционала, предлагаемого на рынке ПО.

На основе проведенного анализа, можно выделить следующие методы синтеза архитектуры ПО ИС: 1) ручной; 2) автоматизированный (в т. ч. с использованием метода диалогового конструирования); 3) автоматический.

В рамках автоматизированного и автоматического методов наиболее широко используются следующие средства формализации: 1) логико-алгебраические спецификации; 2) автоматные модели; 3) графовые модели; 3) ADL-языки; 4) средства онтологического инжиниринга.

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) [24].

Что касается методов проектирования и разработки ПО ИС с использованием наиболее распространенного сборочного подхода, на данный момент существует следующая обобщающая классификация этих методов [24]: 1) модульно-ориентированный; 2) объектно-ориентированный; 3) компонентно-ориентированный; 4) метод генерации; 5) сервисно-ориентированный.

Каждый последующий в этом списке метод является развитием предыдущего. Для реализации данных методов необходимо использовать хранилища готовых решений, компонент повторного использования. При этом указывается на наличие существенных проблем – обеспечение межмодульного интерфейса при сборке ПО ИС [24] и присутствие конфликта между нефункциональными требованиями к компонентам.

Актуальность использования методов ситеза программной архитектуры ИС обусловливается сложными ПрО, которые характеризуются [38]: 1) структурной сложностью и территориальной распределенностью; 2) функциональной сложностью; 3) информационной сложностью; 4) сложностью динамики поведения при высокой изменчивости внешней среды.

Для проектирования современных технически сложных систем широко применяются системы автоматизированного проектирования (САПР). Основными функциями этих систем являются: автоматизация выполнения различных проектных процедур с целью нахождения оптимальных вариантов проектируемого объекта, автоматизация выбора схемы или конструкции, автоматизация составления проектной и технической документации. САПР, ориентированные на конкретную ПрО, используют специальные методы, алгоритмы и программы, оригинальные математические модели, учитывающие специфические качества объектов проектирования. В целях решения проблем повышения эффективности работы ПО ИС

и возможности координации действий программ указывается на необходимость разделения программной системы на управляющий объект (УО) и объект управления (ОУ). УО может служить некая исполнительная система, определяемая как компоновщик, который формирует общий программный код на основе атомарных фрагментов (существующих программных модулей) и алгоритма сборки. Компоновщик дополняет программную архитектуру диспетчером, который контролирует исполнение функций ПО ИС (рисунок 2) [39].

При указанной архитектуре компоновщик может быть сориентирован на повторное использование программных модулей, компонент, инструментального и прикладного ПО. Он может учитывать версионность сборок, а архитектура ИС при этом может быть расширена до включения в её состав нескольких компаний-разработчиков ПО (рисунок 3), что влечет за собой необходимость разработки процессов повторного применения программ [40].

Компания-разработчик может использовать версии ПО, изменять исходный код и, при необходимости, возвращать ПО данной версии с изменениями, указывая, что было модифицировано. В репозитории эти изменения интегрируются с базовой версией ПО. Интегратором является организация, разрабатывающая новые версии ПО. Подобная архитектура хорошо подходит для спиральных и итеративных моделей ЖЦ ПО. В случае спиральной модели осуществимо накопление и повторное использование программных компонентов, моделей и прототипов. Также возможна ориентация на развитие и модификацию ПО в процессе его проектирования.

Основные стадии ЖЦ ПО могут варьироваться в зависимости от выбранной модели ЖЦ (спиральная модель, RUP, MSF), однако процессы ЖЦ

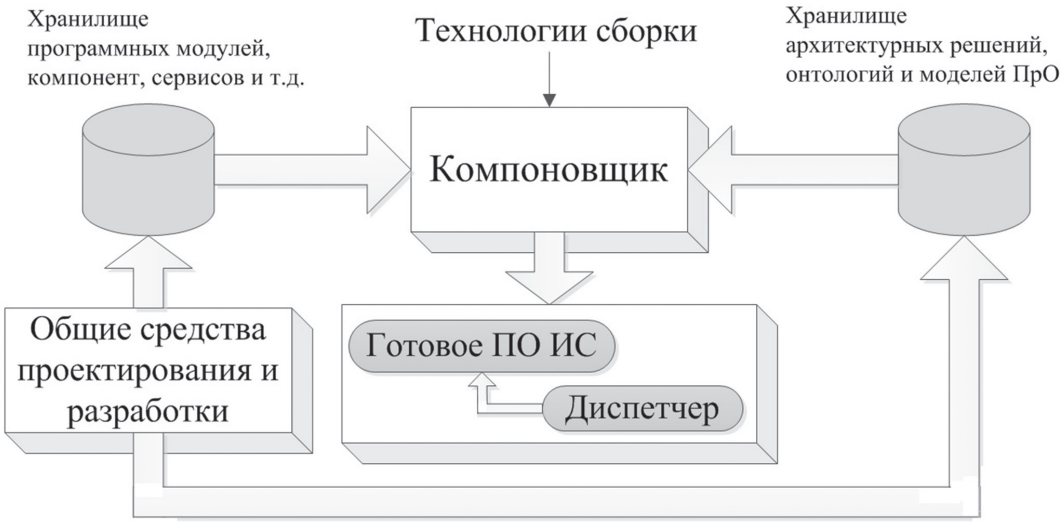

Рис. 2. Концептуальная структура инструментальных средств, реализующих сборку ПО ИС

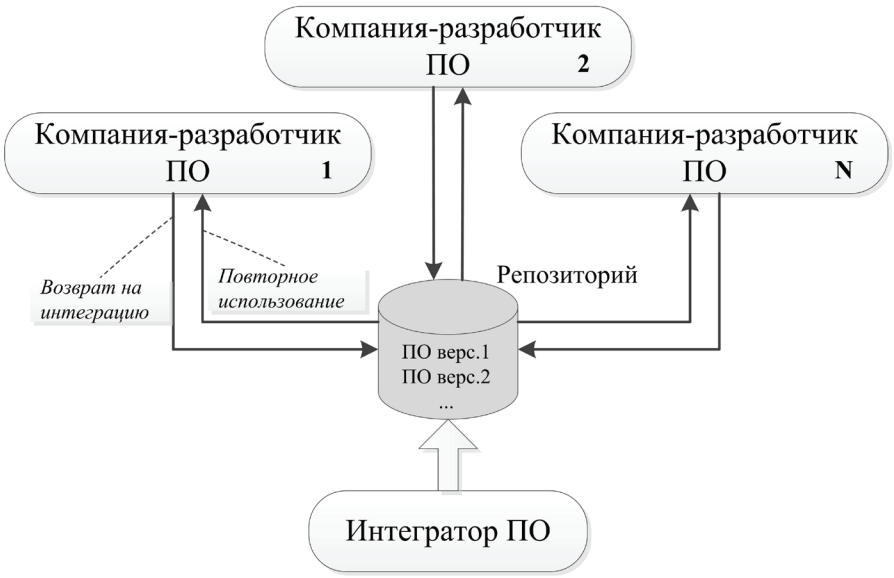

Рис. 3. Менеджмент повторного применения ПО

программных средств регламентируются соответствующими стандартами.

К итеративным моделям относятся наиболее распространенные MSF (Microsoft Solutions Framework) и RUP (Rational Unified Process), которые используют стандарты ISO/IEC [41].

К подвиду итеративных моделей относят модели, применяемые в рамках гибкой методологии разработки ПО (Agile software development): XP (eXtreme Programming), Crystal, FDD (Feature-Driven Development), Scrum, которые могут быть сориентированы как на стандарты CMMI v.1.2 – 1.3 так и на стандарт SPICE (ISO/IEC 15504) [28]. Последние модели (относящиеся к Agile методологии) ориентированы на небольшие компании и штат разработчиков.

Учитывая вариативность представлений стадий ЖЦ в зависимости от моделей, следует ориентироваться на существующие стандарты в этой области для выявления этапов и процессов ЖЦ, которые затрагиваются при применении подходов к оптимизацици ПС.

Согласно выполненному анализу методов синтеза в соответствии со стандартом ISO/IEC 12207:2008 затрагиваются следующие процессы ЖЦ: 1) процесс анализа требований к программным средствам; 2) процесс проектирования архитектуры программных средств; 3) процесс детального проектирования программных средств; 4) процесс конструирования программных средств; 5) процесс комплексирования программных средств. Теория графов нашла широкое применение в рамках указанных процессов. Графовые модели, обладая математической простотой, позволяют описать программную архитектуру и формализуемые задачи ИС. Такие модели являются наглядными и хорошо согласовываются с парадигмами объектно-ориентированного, функционально-ориентированного и компонентно-ориентированного программирования. В связи с тем, что способы представления информации о программной архитектуре ИС, необходимой для генерации программного обеспечения, играют важную роль, подробно рассмотрены графовые модели программной архитектуры.

2. Анализ формальных графовых моделей программной архитектуры информационной системы

Теория графов получила широкое распространение и применяется во многих областях, в частности – для описания программной архитектуры ИС. Графовые модели используются для различных целей в рамках процессов проектирования и разработки ПО: отображения информационных зависимостей между программными компонентами, последовательности выполнения функциональных задач системы, описания версии конфигурации ПО, схемы связей по управлению между элементами программной системы.

Для получения графовой модели программной архитектуры ИС обычно используется информация о заданной ПрО в виде таких моделей ПрО как [42]: 1) информационные модели; 2) модели потоков данных; 3) функциональные модели. Данные модели ПрО берутся за основу при проектировании ПО в рамках существующих технологий проектирования ARIS, IDEF, DFD или UML и являются исходной информацией для формального представления программной архитектуры. Преимуществом графовых моделей архитектуры ПО, базирующихся на формализме теории графов, является способность к визуализации и возможность к автоматизации как процессов проектирования так и разработки программных систем. Теория графов используется для описания архитектуры ПО в граф-ориентированных программных моделях для ВС с параллельной и распределенной архитектурой, а также в целях конфигурирования программных систем [38]. При этом в структуру программмной платформы, реализующей подход к разработке приложений с динамически конфигурируемой параллельной и распределённой архитектурой, включен программный модуль менеджера (диспетчера). Данный модуль осуществляет изменение конфигурации системы во время работы и система не нуждается в перезагрузке. На графовую модель такой программной системы ложится задача описания программных примитивов (программных модулей), реализующих функциональные задачи. В процессе динамического конфигурирования формируется последовательность программных модулей, которые запускаются на выполнение. Эта последовательность исполнения называется конфигурационным планом. Графовая модель представляет собой ориентированный граф, для которого конечному множеству вершин сопоставляются программные модули, а направленным дугам – показатели стоимости и временной задержки передачи данных от одной к другой вершине. Подход к разработке ПО, в основу которого положены граф-ориентированные программные модели, применяется для кластерных вычислений, web-сервисов, компонентно-ориентированных вычислений.

На современном этапе динамическое конфигурирование осуществляется на основе технологии функционального программирования, с использованием функционально-ориентированных языков программирования, таких как Scala [43].

Недостатком указанного граф-ориентированного подхода является то, что при его использовании не учитывается возможность объединения вершин графовой модели для получения супервершин с целью уменьшения связности и сцепления программных модулей, а также упрощения графового представления программной архитектуры ИС. Подобные графовые модели не содержат дополнительной информации, требуемой для их обработки, используются для описания словарей программных классов и представляются кортежем:

$$G = <VC, VA, VR, A, EC, ECO, EA, ER, ERO>,$$

где VC – конструктивные вершины; VA – заменяемые или изменяемые вершины; VR – вершины повторного использования; A – метки; EC – конструктивные дуги; ECO – необязательные конструктивные дуги; EA – изменяемые дуги; ER – дуги повторного использования; ERO – необязательные дуги повторного использования.

Каждой вершине данной графовой модели сопоставляются классы программных модулей. Выделеные на множестве вершин модели подмножество классов повторного использования и подмножество изменяемых классов позволяют описать динамическую часть программной архитектуры. Подобные направленные ациклические графовые модели применяются также в функциональном программировании для автоматизации типизации, поиска соответствующих функций по их заданным аргументам.

Кроме того, на теоретико-множественном уровне представления ИС в качестве основы для процесса синтеза также выступают графовые модели. Примером служит подход, основанный на описании ВП в форме потоков заданий (workflow, WF). В этом случае WF – это ориентированный граф, вершинами которого являются запускаемые задачи, а ребрами – зависимости между задачами по данным и по управлению [44]. Такой подход находит применение для ИС, ориентированных на распределенную вычислительную среду.

Кроме того, среди графовых моделей внимания заслуживают вероятностные модели систем зависимостей. Данные модели на основе графов возникли на стыке многомерного статистического анализа, теории вероятностей, теории графов, теории информации и искусственного интеллекта. Данный класс моделей играет роль строгого языка представления знаний в условиях неопределенности (в частности, в экспертных системах нового поколения) и эффективного аппарата решения разнообразных аналитических задач.

Наиболее привлекательны модели на базе ациклических ориентированных графов (АОГ-модели). Выделяют такие достоинства АОГ-моделей [45]: наглядность, способность отображать причинно-следственные связи и прогнозировать последствия действий (решений), компактное представление систем зависимостей, вычислительная эффективность вероятностного вывода.

Эти свойства обеспечивают эффективное применение таких моделей в медицинской и технической диагностике, социометрическом, эконометрическом и эпидемиологическом анализе, моделировании генетических механизмов, распознавании речи в виде комплекса дисциплин e-Science.

Для задач детального описания архитектуры ПО ИС могут применяться четыре основные графовые модели: граф управления, информационный граф, операционно-логическая история и история реализации [45]. Первые две модели не зависят от входных данных и строятся непосредственно по тексту программы. Две последние модели для своего построения формально требуют слежения за выполнением всех операндов. Сложность построения модели возрастает в порядке указанного перечисления. Все указанные модели существуют для всех программ.

Существует множество сложных научных, инженерных и экономических задач, для решения которых на современных ВС требуется длительное время. При этом количество таких задач постоянно растёт. В крупномасштабных ВС, вероятность потери результатов вычислений очень высока [46]. Исходя из этого, существует серьёзная необходимость обеспечения отказоустойчивого выполнения программ на ВС и в тоже время оптимизации времени работы ВС.

В литературе известен подход к решению этой задачи под названием «backtracking» [47]. Смысл этого метода заключается в использовании «истории» взаимодействий для анализа программы. Вводятся вспомогательные переменные, которые хранят истории взаимодействия по каждому каналу программы. Для хранения историй вводится специальная *историческая переменная* – массив значений, последовательно переданных по соответствующему каналу. А далее при необходимости восстановления этой истории программа обращается к массивам. Кроме принципа backtracking в литературе известен и другой подход, основанный на создании контрольных точек (КТ) вычислений (checkpoint). Подходы к созданию КТ разделяются на реактивный (reactive) и проактивный (proactive) [48]. Наибольшее распространение получил реактивный подход, также называемый *Checkpoint/Restart* или *Rollback/Recovery* [49]. Он предусматривает периодическое создание КТ восстановления, хранящих состояние выполняющейся программы. В случае отказа одного или нескольких

вычислительных узлов (ВУ) любая КТ может быть использована для повторного запуска программы на исправной подсистеме. При этом работа продолжится с момента времени, соответствующего созданию этой КТ. Реактивный подход используется также для балансировки нагрузки ВС и в воспроизводящих отладчиках (Playback Debuggers) [50].

Применение механизма КТ связано с накладными расходами при выполнении параллельных программ (ПП). Данная операция характерна интенсивным использованием узлов ввода-вывода (УВВ), поэтому среднее время создания КТ может быть весьма значительным.

Существует достаточно много средств создания контрольных точек (ССКТ) [51], каждое из которых имеет свои преимущества и недостатки.

Различают две основные схемы взаимодействия ССКТ с защищаемой программой: явная и прозрачная (неявная). Средства создания КТ, построенные на основе явной схемы, позволяют задать ограниченный набор информации, которую необходимо сохранить в КТ. Недостатком явной схемы является необходимость модификации исходного кода, что не позволяет применять её к программам, доступным только в бинарном виде.

Средства создания КТ, построенные на основе прозрачной схемы, осуществляют сохранение КТ незаметно для программы. Недостатком данного подхода является больший объём дискового ввода-вывода информации, так как сохраняется всё пространство памяти.

По классам поддерживаемых программ ССКТ подразделяют на сосредоточенные и распределённые. Сосредоточенные ССКТ обеспечивают отказоустойчивость выполнения одного или нескольких процессов в рамках вычислительного узла ВС. Распределённые ССКТ обычно строятся на базе сосредоточенных и применимы для распределённых и параллельных программ, что делает их важным инструментом организации функционирования ВС.

Для распределённых ССКТ различают координированный и некоординированный подходы. При создании РКТ каждый процесс РП сохраняет свое состояние в КТ. Целостной РКТ называется набор из N локальных КТ, формирующих допустимое состояние программы [49]. Такая РКТ может быть использована для восстановления программы после сбоя. При координированном подходе создание КТ происходит синхронно, что гарантирует целостность РКТ. При некоординированном подходе каждый процесс создает КТ независимо от других. Следовательно, при восстановлении необходимо выполнять поиск целостного состояния программы на основе набора независимых КТ. Последнее вносит дополнительные накладные расходы.

Распределённые ССКТ также можно разделить на универсальные и MPI- ориентированные. Первые ССКТ позволяют создавать РКТ для любых распределённых и параллельных программ, в том числе для различных реализаций модели передачи сообщений (PVM, MPI). Что касается вторых, то существует несколько ССКТ, построенных на базе конкретных реализаций MPI (например, OpenMPI, MVAPICH2). Все они используют пакет BLCR для создания сосредоточенных КТ и реализуют собственные механизмы сохранения графа связей и транзитных сообщений.

Технология автоматного программирования применяется при проектировании такого ПО как системы автоматизации ответственных объектов управления. Стандарт IEC 61499, унифицирующий правила создания распределенных управляющих систем, рекомендует описывать базовые функциональные блоки с помощью конечных автоматов (КА). Выбор в пользу автоматных моделей обусловливается требованиями к организации бесперебойного сохранения и восстановления данных ВП, устранения блокировок и минимизации ошибок [52, 53], что позволяет обеспечивать высокую надежность работы ПО.

Выводы

Выполнен анализ существующих методов структурного синтеза программной архитектуры ИС, в том числе на основе графовых моделей. На основании проведенного анализа установлено, что решение задачи разработки методов и информационных технологий структурного синтеза программной архитектуры, адаптации ПО под изменяющиеся во времени требования конечного пользователя остается актуальным, поскольку на современном этапе достаточно большое число производственных предприятий и компаний (31% – 41%) не удовлетворены уровнем защиты данных для применяемых технологий, а также требуют большой объем доработок исходного ПО, приобретаемого заказчиком. В последнем случае процент кастомизации составляет примерно 11%-25%.

Литература

[1] A Modular Reference Structure for Component-based Architecture Description Languages/Misha Strittmatter, Kiana Rostami, Robert Heinrich [et al.] // ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, September 28, 2015. – Ottawa, Canada, 2015. – P. 36–41.

[2] Ковальчук С. В. Интеллектуальная поддержка процесса конструирования композитных приложений

в распределенных проблемно-ориентированных средах / С. В. Ковальчук, В. Г. Маслов // Изв. вузов. Приборостроение. – 2011. –Т. 54, –№ 10 – С.29–36.

[3] An evolutionary multiobjective optimization approach to component-based software architecture design / R. Li, R. Etemaadi, M.T.M. Emmerich, [at al.] // Congress on Evolutionary Computation (CEC), 5-8 June 2011. – New Orleans, LA.: IEEE, 2011. – P.432–439.

[4] Малышкин В.Э. Параллельное программирование мультикомпьютеров / В. Э. Малышкин, В. Д. Корнеев – Новосибирск: Новосибирский государственный технический университет, 2006. – 452 с.

[5] Левыкин В. М. Модель архитектурного фреймворка ускоренной разработки информационной системы / В. М. Левыкин, М. В. Евланов // Нові технології. – 2013. – № 1-2 (39-40). – С.51–57.

[6] Генератор проектов – средство автоматизации проектирования прикладных информационно-вычислительных систем. / Флёров Ю. А., Вышинский Л. Л., Гринёв И. Л. [и др.] // Автома¬ти¬зация проектирования инженерных и финансовых информацион¬ных систем средствами «Генератора проектов». – М.: ВЦ РАН. – 2010. – С. 3–15.

[7] Инструментальная система ФАКИР / Л. Л. Вышинский, Ю. Д. Прибытков, В. И. Шиленко [и др.] // Известия АН СССР, техническая кибернетика. – 1986. – №3. – С. 6.

[8] [8] Инструментальные средства САПР / Вышинский Л. Л., Гринёв И. Л., Шиленко В. И. [и др.] // Задачи и методы автоматизированного проектирования в авиастроении. – 1991. – С.52–70.

[9] Диалоговая система синтеза многосвязных структур промышленных объектов / [Зайцев И.Д., Кисиль И.М., Вайнер В.Г., Губницкий С.Б.] – Киев: Институт кибернетики. – 1981. – 51 с.

[10] Перевозчикова О. Л. Диалоговые системы / О. Л. Перевозчикова, Е. Л Ющенко; [ин-т кибернетики им. В. М. Глушкова]. – Киев: Наук. Думка, 1990. – 184 с.

[11] Masahiro Sh. Dialog System for Open-Ended Conversation Using Web Documents / Masahiro Shibata, Tomomi Nishiguchi, Yoichi Tomiura // Informatica. – 2009. – № 33. – P.277–284.

[12] Суровцева О. А. Использование потенциала САПР ТП «ТехноПро» для формирования интегрированных комплексов на основе CALS технологий // Состояние и перспективы развития сельскохозяйственного машиностроения: 9-ая междунар. научн.-практ. конф. в рамках 19-й междунар. агропромышленной выставки «Интерагромаш-2016», 2016. – Т. 9. – С. 330–332.

[13] Dan T. An Effort-Based Framework for Evaluating Software Usability Design / Dan Tamir, Carl J. Mueller, Oleg V. Komogortsev // ARPN Journal of Systems and Software. – 2013. –Vol. 3 – № 4. – P.65–77

[14] Корухова Ю. С. Автоматический синтез программ с использованием онтологии прецедентов / Ю. С. Корухова, Н. Н. Фастовец // Программные системы и инструменты: тематический сборник. – Т. 12. – 2011. – С.203–215.

[15] Палагин А. В. Методика проектирования онтологии предметной области / А. В. Палагин, Н. Г. Петренко, К. С. Малахов // Комп'ютерні засоби, мережі та системи. –2011. –№10. – С.5–12.

[16] Зінькович В. М. Онтологічне моделювання предметної області з проблематикою e-Science / В. М. Зінькович // Проблеми програмування. – 2011. – № 3. – С. 30–37

[17] Автоматический синтез системы управления мобильным роботом для решения задачи «Кегельринг» / С. А. Алексеев, А. И. Калиниченко, В. О. Клебан [и др.] // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. – 2011. – № 2 (72). – С.26–31.

[18] Канжелев С. Ю. Автоматическая генерация кода программ с явным выделением состояний / С. Ю. Канжелев // Software Engineering Conference (Russia) – 2006 (SEC (R): матер. конф. 2006. – 2006. – С. 60–63.

[19] Андрианов А. Н. Автоматическая генерация программ для графических процессоров по непроцедурным спецификациям / А. Н. Андрианов, А. Б. Бугеря, Е. Н. Гладкова // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. – 2014. – № 1(3). – С.5–16.

[20] Левыкин В. М. Модель архитектурного фреймворка ускоренной разработки информационной системы / В. М. Левыкин, М. В. Евланов // Нові технології. – 2013. – № 1-2 (39-40). – С.51–57.

[21] Woods E. Using an Architecture Description Language to Model a Large-Scale Information System – An Industrial Experience Report / E. Woods, R. Bashroush // Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 20-24 Aug. 2012 – Helsinki.: IEEE, 2012. – P.239–243.

[22] Coelho K. From Requirements to Architecture for Software Product Lines / K. Coelho, T. Batista // 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), 20-24 June 2011. – Boulder, CO: IEEE, 2011. – P. 282–289.

[23] Агеев Д. В. Параметрический синтез инфокоммуникационных систем с использованием модели многослойного графа / Д. В. Агеев, Фуад Вехбе // СВЧ-техника и телекоммуникационные технологии (КрыМиКо 2013): 23-я междунар. Крымская конф., 8-13 сентября, 2013 г.: тезисы докл. в 2 т. – Севастополь, 2013. – С. 507–508.

[24] Лаврищева Е. М. Software Engineering компьютерных систем. Парадигмы, технологии и CASE-средства программирования / Лаврищева Е. М.– К.: Наук. думка. – 2013. – 283 с.

[25] Фельдман Л. П. Эффективность реализации параллельных вычислений для кластерных систем на базе интерфейса MPI / Л. П. Фельдман, И. А. Назарова // Наукові праці Донецького національного технічного університету. Серія : Інформатика, кібернетика та обчислювальна техніка. – 2016. – № 1. – С. 136–141.

[26] Mohammadkhanli L. Ranking Approaches for Cloud Computing Services Based on Quality of Service: A Review / Leyli Mohammadkhanli, Arezoo Jahani //ARPN Journal of Systems and Software. – 2014. – Vol. 4. – № 2. – P.55–62.

[27] Сергеев Ю. Управление доступом к виртуальной инфраструктуре с помощью продукта HyTrust / Ю. Сергеев // Jet Info Информационный бюллетень. – 2012. – №3 (224). – 44 с.

[28] Веденеев В. С. Применение экстремального программирования при разработке научных приложений / В. С. Веденеев, И. В. Бычков // Математические структуры и моделирование. – 2014. – №. 4 (32). – С.180–184.

[29] Amodeo E. Learning Behavior-driven Development with javascript + Code / Enrique Amodeo. – Birmingham: Packt Publishing, 2015. – 392 p.

[30] Smart J. F. BDD in Action: Behavior-driven development for the whole software lifecycle / John Ferguson Smart / Publisher: Manning Publications, Shelter Island, NY, 2015. – 384 p.

[31] Marciuska S. Automated Feature Identification in Web Applications / S. Marciuska, C. Gencel, P. Abrahamsson // International Conference on Software Quality. – Springer International Publishing, 2014. – P. 100–114.

[32] Customer-Induced Interactions And Innovation In Professional Services: The Case Of Software Customisation / M. Schaarschmidt, W. Gianfranco, B. Matthias, V. K. Harald // International Journal Of Innovation Management. – 2015. – P.1–38. – Режим доступа: https://www.academia.edu/21938879/Customer-Induced_Interactions_ and_Innovation_in_Professional_Services_The_Case_of_Software_Customization/

[33] Nwankpa J. K. Real Options and Subsequent Technology Adoption: An ERP System Perspective / J. K. Nwankpa, Y. Roumani // System Sciences (HICSS): 48th Hawaii International Conference. – IEEE, 2015. – P. 5020–5027.

[34] Learning and Evolution in Dynamic Software Product Lines / Amir Molzam Sharifloo, Andreas Metzger, Clement Quinton, [at al.] – 2016. – 8 p. – Режим доступа: https://hal.archives-ouvertes.fr/hal-01280837

[35] Baresi L. Service-oriented dynamic software product lines / L. Baresi, S. Guinea, L. Pasquale // Computer. – 2012. – Т. 45. – №. 10. – P. 42–48.

[36] 2015 ERP report [Електронный ресурс] // Panorama Consulting Solutions, LLC. – Denver, Colorado, 2015. – Режим доступа: http://panorama-consulting.com/resource-center/2015-erp-report/

[37] 2016 Report on ERP systems and enterprise software [Електронный ресурс]// Panorama Consulting Solutions, LLC. – Denver, Colorado, 2016. – 32 p. – Режим доступа: http://panorama-consulting.com/resource-center/2016-erp-report/

[38] Вендров А. М. Современные технологии создания программного обеспечения. Обзор / А. М. Вендров // Jet Info. –2004. –№4 (131). – С.3–32.

[39] Lifecycle Management of Open-Source Software in the Public Sector. A Model for Community-Based Application Evolution / Ju. Kääriäinen, P. Pussinen, T. Matinmikko, T. Oikarinen // ARPN Journal of Systems and Software. – 2012. – Vol. 2. – № 11. – P.279-288.

[40] Брагина Т. И. Сравнительный анализ итеративных моделей разработки программного обеспечения / Т. И. Брагина, Г. В. Табунщик // Радіоелектроніка, інформатика, управління. – 2010. – Вып. № 2 (23). – С.130–139.

[41] Чайников С. И. Методы и алгоритмы априорной оценки параметров вычислительных процессов: автореф. дис. на соиск. уч. степени канд. техн. наук: спец. 05.13.01 «Техническая кибернетика и теория информации» / С. И. Чайников.– Харьков: ХИРЭ – 1983.–16 с.

[42] Casadei R. Towards Aggregate Programming in Scala / R. Casadei, M.Viroli // First Workshop on Programming Models and Languages for Distributed Computing. – ACM, 2016. – P. 5.

[43] Князьков К. В. Предметно-ориентированные технологии разработки приложений в распределенных средах /К.В.Князьков, А.В.Ларченко // Изв. вузов. «Приборостроение». – 2011. – Т. 54 – № 10 – С.36–43.

[44] The TETRAD Project: Constraint Based Aids to Causal Model Specification / Richard Scheines, Peter Spirtes, Clark Glymour [et al.] // Multivariate Behavioral Research. – 1998. – Vol. 33 –№ 1. – P.65–118.

[45] Воеводин В. В. Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – Санкт-Петербург.: БВХ-Петербург, 2002. – 608 с.

[46] Поляков А. Ю. Оптимизация времени создания и объема контрольных точек восстановления параллельных программ / А. Ю. Поляков, А. А. Данекина //Вестник СибГУТИ. –2010. – №2 –С.87–100.

[47] Антонов В. В. Построение формальной модели предметной области с применением нечеткой кластеризации / В. В. Антонов, Г. Г. Куликов, Д. В. Антонов // Уфа: УГАТУ. – 2011 –Т. 15 –№ 5 (45). – С. 3–11.

[48] Proactive fault tolerance for HPC with Xen virtualization / A.B. Nagarajan, F. Mueller, C. Engelmann, S. L. Scott // ICS 2007: proc. of the 21st Annual International Conference on Supercomputing. – ACM, New York, 2007. – P. 23–32.

[49] A survey of rollback-recovery protocols in message-passing systems / Elnozahy E. N., Alvisi L., Wang Y. M. [at al.] // ACM Computing Surveys. – 2002. – Vol. 34 – №3. – P. 375–408.

[50] Proactive process-level live migration in HPC environments / C. Wang, F. Mueller, C. Engelmann, S. L. Scott // In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC). – 2008. – P.1–12.

[51] The design and implementation of checkpoint/restart process fault tolerance for Open MPI / Hursey J., Squyres J.M., Mattox T.I. [at al.] // In Proceedings of the 21st IEEE International Parallel and stributed Processing Symposium (IPDPS). – IEEE Computer Society. – 2007. – Vol. 3– №26 – P.1–8.

[52] Построение автоматных программ по спецификации с помощью муравьиного алгоритма на основе графа мутаций / Чивилихин Д. С., Ульянцев В. И., Вяткин В. В. [и др.] // Научно-технический вестник информационных технологий, механики и оптики. – 2014. – № 6 (94). – С. 98–105.

[53] Шелехов В. И. Язык и технология автоматного программирования // Программная инженерия. – №4. – 2014. – С. 3–15.

Поступила в редколлегию 14.05.2019