УДК 004.942

А. П. Соколов, В. О. Голубев
Московский государственный технический университет им. Н. Э. Баумана

Система автоматизированного проектирования композиционных материалов. Часть 3. Графоориентированная методология разработки средств взаимодействия пользователь–система*

Несмотря на активное развитие средств автоматизации, прикладное наукоемкое ПО, включая системы инженерного анализа и автоматизации проектирования, полноценные облачные платформы продолжают оставаться доступными лишь ограниченному кругу специалистов, что обусловлено их дороговизной и, как правило, сложностью применения. Вместе с тем, потребность в таких системах для ученых, инженеров, обучающихся только возрастает. В статье представлен оригинальный графоориентированный метод организации вычислительных процессов в САПР, а также основанная на его применении методология разработки специализированных модулей расширения, реализованная для добавления новых функциональных возможностей в представляемую САПР КМ и позволяющая описывать логику работы пользователя в системе для решения конкретной прикладной задачи (вычислительной, задачи проектирования и пр.). Проведен анализ существующих программных решений в области организации сложных вычислительных процессов в распределенной вычислительной среде и представлен перечень программных средств, которые необходимы для организации соответствующей программной инфраструктуры. Обоснована актуальность работы в выбранном направлении. Представленные метод и методология призваны упростить процессы как разработки новых возможностей, так и работы в сложном наукоемком программном обеспечении. Представлены некоторые детали программной реализации отдельных элементов разрабатываемой методологии, а также результаты тестирования разработанных программных средств, обеспечивающих взаимодействие пользователя с системой в процессе обхода тестовой графовой модели. Созданные программные средства позволяют визуализировать текущий статус обхода графовой модели в реальном масштабе времени. Приведен пример графовой модели верхнего уровня абстракции, реализующей методологию автоматизированного проектирования КМ в рамках разрабатываемой САПР КМ, построенной с использованием представленной методологии.

Технологии разработки инженерного программного обеспечения, графоориентированный подход, описание бизнес-логики, реактивное программирование, low-code-разработка, разработка систем инженерного анализа, разработка модулей расширения, технологии распределенных вычислений, потоко-ориентированные системы, инструменты e-science, автоматизированное проектирование композиционных материалов

Применение наукоемкого программного обеспечения (ПО), в частности работа со сложными САПР, системами инженерного анализа (CAE) и пр., для пользователя обычно сопряжено с требованием глубоких знаний предметной области решаемых задач. Многие вычислительные задачи, которые требуется решать в процессе проектирования сложных технических объектов, как правило, предполагают многочисленные процедуры подготовки, обработки и постобработки данных [1]. Каждая из этих процедур, в контексте применяемой САПР включает необходимость запуска различных специальных функций, назначение и принципы работы которых пользователь вынужден предварительно изучить на основе документации по системе. Вместе с тем, разнообразность актуальных прикладных задач ведет к новым требованиям в отношении соответствующей САПР, что в итоге приводит к необходимости многочисленных изменений такого ПО, добавлению новых функциональных возможностей и обязательно отражается на документации. Такое положение дел еще больше усложняет процессы проектирования, создания, сопровождения и применения подобного программного обеспечения.

* Окончание. Начало см. в Изв. СПбГЭТУ «ЛЭТИ» 2020. № 8–9, 10.

Для успешного внедрения и эффективного использования наукоемкого ПО, к которому относится и рассматриваемая САПР композиционных материалов (САПР КМ), разработчик должен учитывать:

– применение такого ПО без документации, как правило, невозможно;

– необходимость документации у пользователя такого ПО тем большую, чем сложнее (или чем хуже) организована работа в системе;

– чем больше нужда в документации, тем менее привлекательной для пользователя становится такая система (большинство пользователей не готовы изучать документацию вовсе).

Принципы обновления документации определяются методами ее изначального построения: она может быть разработана вручную или с использованием технологий автоматического построения – возможно, за счет интерпретации исходных кодов и анализа стандартно оформленных комментариев, что реализовано, к примеру, в ПО doxygen; могут применяться технологии генерации кода на основе шаблонов (template-based code generation, TBCG) – например, в ПО Acceleo, Xpand, XSLT, Velocity и др. (более подробно данная тематика представлена в [2]). Аналогичные технологии активно используются крупными производителями программного обеспечения («Microsoft», «Oracle», SAP AG и пр.).

Применение программных средств автоматического построения документации часто обоснованно для построения справочных материалов, например для разработчиков – описания программных интерфейсов, архитектур, иерархий классов, методов и прочая техническая информация. Возможности применения всего этого конечным пользователем с целью изучения принципов использования ПО для решения конкретных прикладных задач существенно ограничены. Вместе с тем, документация, разработанная вручную, учитывающая, например, языковые особенности восприятия целевой аудитории, включая примеры практического применения, не может «похвастаться» удобством ее обновления, актуальностью (соответствием текущей версии ПО), требует существенных усилий для ее разработки.

Современный процесс разработки программных систем предполагает применение многочисленных вспомогательных систем: сред разработки, систем контроля версий, систем непрерывной интеграции, генерации документации и другого (т. н. CASE (computer-aided software engineering)) инструментария [3]. Тем не менее, их недостаточно для создания конкурентоспособных на мировом рынке САПР.

Утверждение 1. Разработчик наукоемкого ПО должен добиваться снижения необходимости изучения документации при практическом применении такого ПО (возможно даже бóльше, чем при разработке ПО общего назначения).

Обновляя ПО, разработчик должен успевать обновлять «полезную для пользователя» документацию, а пользователь – естественным образом «воспринимать» новые возможности как что-то само собой разумеющееся. В большинстве случаев это осуществимо только в автоматизированном режиме, что, в свою очередь, возможно только при соответствующих условиях разработки и последующего использования системы.

Утверждение 2. Одним из возможных путей к упрощению процесса применения и изучения наукоемкого ПО служит исключение необходимости его изучения за счет автоматизации самой процедуры его использования. Такая автоматизация предполагает: а) формализацию метода организации вычислительных процессов в автоматизированной системе; б) определение классов прикладных задач, поддерживаемых в системе, и сопоставление с каждым классом формального описания метода решения; в) ограничение доступа к функциональным возможностям системы, обеспечивающим решение отдельных подзадач.

Классификация задач и функций в контексте описываемой САПР КМ была представлена в [4] и [1]. Далее будем использовать термин «решатель» в отношении программного обеспечения, предназначенного для вычислительных задач одного класса [1].

Рассмотрим более детально процесс взаимодействия исследователя с наукоемкой системой. Для эффективной работы с подобной системой исследователю необходим графический пользовательский интерфейс, в котором модель организации вычислений может быть представлена визуально. Большие перспективы в автоматизации процесса решения сложных задач перед исследователем открывает возможность прямого взаимодействия с вычислительной моделью: остановка вычислительного процесса на определенном этапе, изучение обрабатываемых данных, просмотр истории изменения обрабатываемых данных, воз-

врат к определенному этапу вычислений, ввод дополнительных параметров на определенной стадии вычислений и т. д. О необходимости подобных решений писал еще Б. Маккормик [5].

Утверждение 3. Многие известные методики предполагают организацию вычислительных процессов в графовой форме. Например, применяют: диаграммы потоков данных (DFD), граф-схемы, конечные автоматы, диаграммы перехода состояний.

Такое описание позволяет выделять структурные единицы приложения в виде функций или подпрограмм и связывать их между собой в определенной последовательности. Продолжением идей структурного описания проектируемого приложения можно считать объектный подход, в котором стали применять методы объектно-ориентированного программирования, что впоследствии привело к методологии компонентно-ориентированной разработки (component-based development, CBD). Задолго до CBD широкого применения М. Д. Макилроем [6] были изложены некоторые ее парадигмы, среди которых ключевой становится унификация программного интерфейса для интеграции программных модулей в сложную систему.

Важной вехой в области унификации процесса организации обработки данных можно считать создание универсального языка моделирования UML [7]. Для визуализации процессов используются диаграммы последовательностей, состояний, деятельности. Вместе с тем, UML не стал общепринятым средством, упрощающим разработку вычислительных методов, и тем более средством организации сложных вычислительных процессов.

Один из способов организации и выполнения поточной обработки данных, реализующий концепцию CBD, описан Р. Д. Бьорнсоном и С. Б. Уэстоном [8]. Базовой идеей служит организация вычислительного процесса в форме диаграммы потоков данных (data-flow diagram, DFD), когда узлам диаграммы ставятся в соответствие стандартизованные компоненты, осуществляющие промежуточную обработку данных. Для создания и редактирования диаграммы предусмотрен графический интерфейс, в котором можно определять связи между компонентами посредством указания точек входа и выхода обрабатываемых данных.

Метод разработки программного обеспечения при помощи проектирования моделей, не зависящих от предметной области, запатентовали Д. Тавискот и Д. Макмастер в 2015 г. [9]. Метод заключается в построении множества UML-диаграмм, описывающих процессы обработки данных, на основе интерпретации которых и с использованием шаблонов далее генерируется исходный код приложения. Подобная техника может применяться и для создания программного обеспечения для реализации сложных вычислительных методов решения инженерных задач. Вместе с тем, представленный подход не специализирован для таких задач и, представляется, успешно может применяться только системными архитекторами.

Подход к построению моделей обработки данных вычислительных экспериментов предложили российские исследователи Д. В. Леонтьев, В. Г. Тарасов, И. Д. Харитонов [10]. Для построения моделей используются сети Петри. Модель вычислительного эксперимента состоит из моделей вычислительного и управляющего процессов. Строятся эти модели раздельно. Построение модели вычислительного процесса происходит в два этапа: сначала пользователь формирует дерево событий вычислительного эксперимента, далее на его основе происходит автоматическое построение модели вычислительного процесса. Модель управляющего процесса строится на основе предварительно определенных шаблонов реакции на события. К важным особенностям подхода относится отсутствие необходимости в ручной подготовке исходного кода алгоритма решения вычислительной задачи: реализация алгоритма осуществляется за счет автоматической интерпретации полученных моделей. Подход в определенной степени позволяет строить модели обработки данных пользователям с минимальной специальной подготовкой, реализует гибкую (легко сопровождаемую) структуру вычислительного эксперимента, обеспечивает возможности отслеживания текущего статуса эксперимента за счет наличия модели управляющего процесса.

Замечание 1. В контексте задачи формализации метода организации вычислительных процессов следует учитывать, что для решения конкретной прикладной задачи может потребоваться выполнить отдельные вычислительные процедуры на удаленной высокопроизводительной вычислительной системе. Обычно при этом применяют методы параллельные вычислений, распределенные вычисления, GRID-технологии, готовые облачные платформы.

GRID-системы применяются учеными для вычислительных задач, когда производительности кластеров, состоящих из мощных серверов недостаточно [11]. Р. К. Алекперов в своем исследовании [12] рассмотрел организацию распределенных вычислительных сред на основе GRID-технологии и проанализировал специфические особенности этих задач. В своей работе он привел практические примеры, когда распределенные вычисления помогли решить прикладную задачу за несколько недель, тогда как при использовании персональных компьютеров этот расчет потребовал бы десятки лет вычислений. В работе ученых З. Фаркаса и П. Кацука описана разработка графического пользовательского интерфейса для взаимодействия с различными видами GRID-сетей [13]. Графический интерфейс включает визуализацию структуры используемой GRID-сети в форме графа и позволяет отображать на данной структуре поток выполняемых работ (workflow), тем самым обеспечивая возможности взаимодействия с вычислительной системой и визуализацию состояния выполняемых задач. В системе реализован редактор, который позволяет изменять структуру вычислительной системы. Взаимодействие с различными видами GRID-сетей реализуется с использованием программного обеспечения промежуточного уровня. Для каждой конкретной архитектуры GRID-сети должны быть разработаны специальные скрипты, которые обеспечивают: преобразование действий пользователя в графическом интерфейсе на язык конкретной архитектуры, прием и интерпретацию ответов от GRID-сети с соответствующей визуализацией для пользователя.

К самым популярным программным системам для запуска вычисления и развертывания GRID-сетей относится BOINC, которая используется во многих научных проектах.

Несмотря на большую производительность, развертка GRID-сети – не самое лучшее решение для проектов с ограниченными ресурсами и задачами, не требующими множества часов вычислений. Более рациональным вариантом может оказаться задействование облачных платформ. Облачные платформы призваны скрывать от исследователей системный уровень обработки сложной вычислительной задачи, включающий процессы распределения вычислительной нагрузки в многопроцессорной среде, выбор наименее нагруженных вычислительных узлов для проведения вычислений, прием и отправку сообщений от одного вычислительного узла другому и пр., предоставляя вычислительные ресурсы в наиболее привычном для использования виде: доступ по протоколу SSH, доступ к вспомогательным сервисам мониторинга вычислительных процессов и т. д.

Уже традиционно доступ к облачным платформам предоставляется одним из трех методов: «инфраструктура как услуга», когда пользователю предоставляется расширенный доступ к базовому ПО организации облачных вычислений (infrastructure as a service (IaaS)); «платформа как услуга», когда пользователю предоставляется расширенный доступ к уже настроенной инфраструктуре облачных вычислений (platform as a service, PaaS), и, наконец, «система как услуга», когда пользователю предоставляется доступ к специализированному ПО – как правило, посредством веб-интерфейса или API-поставщика (system as a service, SaaS). Хорошо известны облачные платформы широкой области применения уровня инфраструктуры (IaaS) с открытым исходным кодом – системы Eucalyptus, Nimbus, OpenNebula, CloudStack, OpenStack, а также с закрытым исходным кодом – Amazon EC2, Windows Azure, Google Compute Engine и др. [14].

Многие облачные вычислительные платформы ориентированы в основном на решение задач машинного обучения: Orange [15], TensorFlow [16]. Среди закрытых разработок следует упомянуть проект вычислительной платформы российской компании «Яндекс» – Нирвана [17], который позволяет проводить вычисления, не завязанные на конкретную предметную область. Подобный подход позволяет использовать вычислительную платформу для различных задач.

Известны и специализированные платформы для запуска решателей от всемирно известных производителей программного обеспечения – «Siemens», ANSYS, MSC и др. Подобные решения позволяют исследователю сосредоточиться на решении прикладных задач, а представитель сервиса берет на себя обязательства по предоставлению необходимых вычислительных мощностей с предустановленным программным обеспечением.

Замечание 2. Применение открытых облачных платформ удобно, так как снимает бóльшую часть ответственности по инфраструктурному обеспечению вычислительной системы с конечного пользователя, однако вместе с тем может быть недопустимо для конкретной организации из соображений конфиденциальности обрабатываемых данных и получаемых результатов.

Все представленные подходы и решения предполагают описание комплексного вычислительного процесса в форме ориентированных графов состояний [15], сетей Петри [10] или диаграмм потоков данных и предоставления пользователю соответствующего графического интерфейса. Подобные возможности также реализуются за счет применения ПО промежуточного уровня, которое обеспечивает совместное функционирование готовых программ и библиотек пользователя, организуя их в единый поток работ, не скрывая факт их выполнения на удаленных вычислительных системах (в облачной среде).

Графическая форма представления сложного вычислительного процесса обеспечивает наглядность его разработки и доработки, позволяет отслеживать процессы выполнения (workflow monitor) и управлять ими, включая возможности визуализации обрабатываемых данных. Многие из представленных решений включают в свой состав программные средства, обеспечивающие указанные возможности, включая отображение деталей и историю выполнения процессов; позволяют остановить, прекратить или возобновить исполнение и т. д. Подобные механизмы реализуются с помощью создания параллельного управляющего процесса, который может обмениваться сообщениями с основным исполняющим процессом [10], [13].

При разработке программных средств поддержки графического описания сложных вычислительных процессов должен быть реализован «прозрачный» механизм удаленного запуска отдельных вычислительных процессов. На рис. 1, *а* представлена простейшая графовая модель [18], для которой предполагается, что функции перехода F_1 и F_2 должны быть выполнены в вычислительной системе Node 1, а функция перехода F_3 – в Node 2 (рис. 1, *б*). Для реализации указанных задач должен быть выполнен целый комплекс дополнительных операций (рис. 1, *б*).

В 1970–80-х гг. появился ряд реализаций подхода удаленного запуска процедур (RPC – Remote Procedure Call, документы RFC-1057, RFC-1057). Подобные технологии предлагали подход к запуску удаленных вычислений таким образом, чтобы это было похоже на вызов обычной процедуры в языках программирования. Среди основных можно выделить DCOM, CORBA, SOAP (XML-RPC) DCOM и др. На сегодняшний день существуют более современные их аналоги – Thrift [19], Finagle [20], gRPC [21].

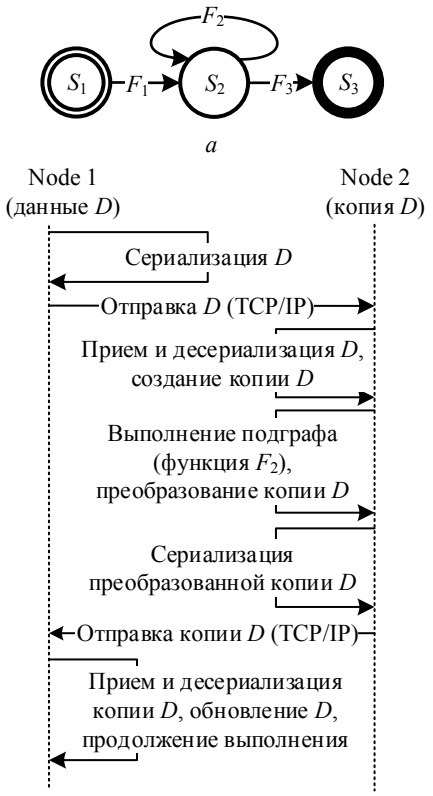


Рис. 1

Нельзя не упомянуть архитектурный подход к взаимодействию компонентов распределенного приложения в сети – REST (REpresentational State Transfer) [22], который очень хорошо себя зарекомендовал как способ обмена данными и командами в клиент-серверной архитектуре. Поскольку REST использует такие стандарты, как HTTP, URL, JSON и XML (eXtensible Markup Language), послать запрос на удаленный запуск процедуры на сервере не представляет большого труда.

Реализации RPC предлагали не только способы удаленного вызова, но и технологии сериализации-десериализации данных и даже шифрования, однако для передачи больших данных, используемых учеными и исследователям, производительность была недостаточной. Для научных наборов данных, имеющих большой объем, – данных, получаемых от спутников, или численных моделей климата, погоды и океанов, – были разработаны специальные бинарные стандарты сериализации, например HDF (Hierarchical Data Format) [23] и netCDF (Network Common Data Form) [24].

Для передачи разнородных данных по сети особую популярность приобрел язык разметки XML, однако программная обработка XML может оказать-

ся неоправданно затратной по сравнению с работой с данными, имеющими более простую структуру. В таком случае разработчики рассматривают средства, изначально ориентированные на данные, – INI (Initialization file), YAML (YAML Ain't Markup Language), JSON (JavaScript Object Notation).

Утверждение 4. В результате проведенного анализа можно резюмировать, что для формализации метода организации вычислительных процессов в рамках наукоемкого ПО, в том числе в рамках САПР, необходима реализация многочисленных вспомогательных программных средств, к которым относятся:

1) средства интерпретации графового описания сложных вычислительных процессов;

2) средства запуска процессов обработки данных на удаленных вычислительных системах;

3) средства поддержки универсальных форматов данных (INI, YAML, JSON, XML) включая инструменты их сериализации и десериализации;

4) средства обеспечения процессов установки соединения, приема и передачи данных и команд по определенному протоколу;

5) графический пользовательский интерфейс, предоставляющий средства визуализации и редактирования графического представления всего вычислительного процесса;

6) средства управления процессом, включающие возможности остановки вычислительного процесса на определенном этапе, изучение обрабатываемых данных, просмотр истории изменений обрабатываемых данных, возврат к определенному этапу вычислений, ввод дополнительных параметров на определенной стадии вычислений и т. д.;

7) средства автоматического распределения нагрузки на CPU и GPU [25];

8) средства защиты от сбоев, резервирования данных расчетов, максимально быстрого восстановления результатов вычислений (например, Microsoft Cluster Server, Beowulf, Condor, MOSIX, EnFuzion, PBS [26]); некоторые из таких средств представлены наборами дополнений к ядру операционной системы.

Над решением подобных задач работали многие исследователи и производители программного и аппаратного обеспечения. Их реализация в полном объеме доступна, как правило, крупным производителям ПО. Вместе с тем, конечный исследователь, будучи ограниченным в ресурсах (как финансовых, так и временных), вынужден выбирать между решениями различных ценовых диапазонов, обладающими различными функциональными возможностями – ориентированными на решение прикладных задач или нет, требующими дополнительного обучения или нет, и т. д. В указанных условиях, несмотря на большое количество технологий, одним из самых практичных до сих пор считается простейший удаленный доступ к вычислительному кластеру и запуск на его мощностях необходимых вычислений, с использованием известных протоколов SSH, FTP и SFTP. Однако такой подход часто неприменим при необходимости решения прикладных задач.

Резюмируя, можно заключить, что применение графических форм представления сложных процессов обработки данных не ново и активно применяется многими известными производителями программного обеспечения, в том числе облачных вычислительных платформ.

Актуальность работ в выбранном направлении обусловлена активным развитием средств автоматизации, вместе с тем наукоемкое ПО, включая системы инженерного анализа, системы автоматизации проектирования, полноценные облачные платформы, продолжает оставаться доступным лишь ограниченному кругу специалистов. Вместе с тем, потребность в автоматизированном решении прикладных задач лишь возрастает.

Настоящая статья завершает цикл из трех статей о базовых принципах, которые легли в основу разрабатываемой системы автоматизированного проектирования композиционных материалов (САПР КМ). Первая часть была посвящена концепциям, архитектурным особенностям и описанию платформы разработки САПР КМ [4]. Во второй части были представлены основы вычислительной подсистемы и принципы организации распределенных вычислений с применением графоориентированного подхода [1], теоретические основы которого были опубликованы ранее [18]. Цель настоящей статьи – представить *метод организации вычислительных процессов* в САПР, а также *методологию разработки специализированных модулей расширения*, реализованную для добавления новых функциональных возможностей в САПР КМ и позволяющую описывать логику работы пользователя в системе для решения

конкретной задачи (вычислительной, задачи проектирования и пр.).

Метод организации вычислительных процессов в САПР. В соответствии с терминологией графоориентированного подхода [18] функции-обработчики f_i (рис. 2) вполне могут реализовывать не только автоматическое преобразование данных из одного состояния в другое, но также и обеспечивать запрос дополнительных данных от лица, принимающего решение, (ЛПР) в процессе автоматического обхода соответствующей графовой модели.

В основе предлагаемого авторами метода организации вычислительных процессов в САПР лежит обобщенное представление об интерпретации понятия «графовая модель», с помощью которой, очевидно, может быть описан не только алгоритм реализации сложного вычислительного метода [18], [27], но также, к примеру:

1) логика работы в системе пользователя с определенной ролью (рис. 2): h – функция-селектор, связанная с состоянием S_2;

2) логика решения сложной прикладной задачи из фиксированного класса задач [1] (например, задача автоматизированного проектирования композиционного материала с заранее определенными упругими характеристиками), для которой характерно проведение экспериментальных исследований, сравнение и сохранение результатов, а также синхронное участие различных специалистов;

3) последовательность состояний результатов расчетов, характеризующих степень доверия к хранимому результату (рис. 3), где использованы обозначения: S_1 – результат получен численно; S_2 – проведен качественный анализ; S_3 – результат получен численно, проведено количественное сравнение с аналитическим результатом; S_4 – результат получен численно, проведено количественное сравнение с результатом эксперимента; S_5 – результат получен численно, проведено количественное сравнение с результатом эксперимента, аналитическим результатом и численным результатом, полученным эталонным методом; h – функция-селектор, связанным с состоянием S_1, где числами, поставленными в соответствие с исходящими ребрами, обозначены приоритеты выполнения;

4) последовательность состояний утверждения документа (активно используется в системах документооборота).

На рис. 2 и 3 с помощью h, p_j, f_i обозначены функции – селекторы, предикаты и обработчики соответственно [1], [18]. Числами, поставленными в соответствие ребрам, исходящим из узлов, соответствующих состояниям S_1 (рис. 3), S_2 (рис. 2), обозначены приоритеты выполнения (для машины с одним процессором, имеющим одно ядро) соответствующих функций перехода в случае, если в результате работы соответствующей функции-селектора h обход должен быть проведен по нескольким из ребер (см. определение функции-селектор в [1]). Для графовой модели, представленной на рис. 2, функции-обработчики f_2, f_4, f_6 инициируют создание диалоговых окон W_2, W_4, W_6, запрашивающих у пользователя ввод дополнительных данных. Для реализации диалоговых окон следует использовать методики их динамического построения, что можно реализовать за счет применения универсальных текстовых форматов данных (INI, YAML, JSON) и их последующей интерпретации [28].

Методология разработки средств взаимодействия пользователь–система в САПР КМ. Разработка модулей расширения в САПР КМ, основанных на графовых моделях и позволяющих запрашивать дополнительный ввод данных, предполагает следующие процедуры:

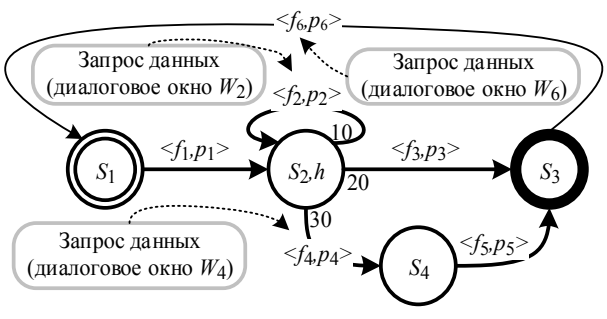


Рис. 2

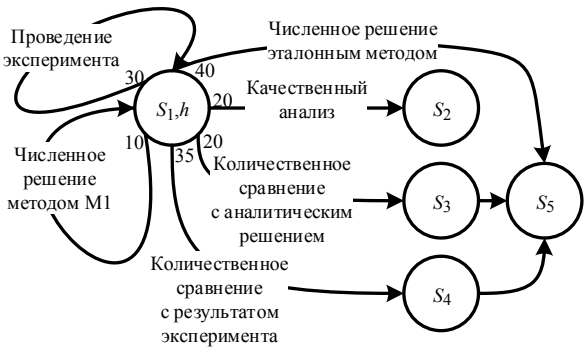


Рис. 3

1. Создать графовую модель G, описывающую логику работы пользователя в системе *для решения задач из фиксированного класса* в соответствии с графоориентированным подходом [27]. Модель следует зарегистрировать в системной базе данных САПР КМ, формально определив ее с использованием языка aDOT [29] и разместив файл в репозитории графовых моделей системы.

2. Связать графовую модель с функциями-селекторами h_k, функциями-предикатами p_j и функциями-обработчиками f_i в рамках определения графовой модели в формате aDOT.

3. Разработать файлы, определяющие диалоговые формы W_l запросов дополнительных данных с использованием языка aINI, создать на их основе функциональные компоненты системы [28], которые следует использовать в качестве функций-обработчиков, связываемых с теми ребрами ранее определенной графовой модели, выполнение которых требует участия лица принимающего решение.

4. Зарегистрировать новый функциональный компонент C системы в системной базе данных, определив для него в качестве атрибутов связанную графовую модель G и базовый шаблон исходных данных также в формате aINI.

Особенности программной реализации. Реализация представляемой методологии в общем случае предполагает разработку ряда вспомогательных программных средств (см. утверждение 4).

Средства интерпретации графовых моделей были реализованы ранее на двух языках программирования C++ (библиотека comsdk) и Python (библиотека pycomsdk) [1].

Для создания базовой реализации в контексте разрабатываемой САПР КМ требовалось решить множество технических задач по дополнению существующих инструментов, среди прочих: а) обеспечение возможности приема, передачи и интерпретации сообщений к графовой модели; б) проектирование и разработка плагина (модуля расширения) визуализации процесса решения вычислительной задачи на основе его графовой модели (было осуществлено в рамках проекта разработки веб-клиента к САПР КМ comwpc [4]); в) проектирование и разработка нового типа плагинов, позволяющего создавать новые функции системы, основанные на некоторой графовой модели, зависящей от других функций системы, в том числе функциональных компонентов [1], и в том числе функциональных компонентов, основанных на применении формата aINI [28]; г) проектирование и разработка интегрируемого контекстного меню для запуска решателя, связанного с объектом системы.

К моменту подготовки настоящей статьи были реализованы пп. а и б, пп. в и г станут предметом будущих работ.

Прием, передача и интерпретация запросов к графовой модели. Исходя из анализа научных работ и известных программных разработок, был предложен подход к созданию архитектуры подсистемы приема, передачи и интерпретации запросов к графовой модели. Была использована идея разделения запуска процесса вычислений на два параллельно выполняемых процесса: управляющий и исполнительный (рис. 4). Функции управляющего процесса: порождение исполнительного процесса; прием запросов и выдача ответов; управление вычислениями. Функции исполнительного процесса: обход графовой модели; передача статуса решения. Такой подход позволяет не усложнять логику обхода графовой модели, поскольку обязанности по коммуникации и управлению вычислениями берет на себя управляющий процесс.

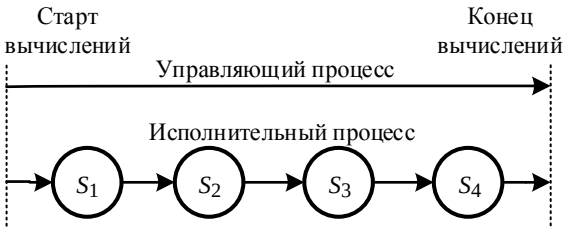


Рис. 4

Такой подход позволяет не только получать статус текущего решения, но также обеспечивает реализацию процессов отладки, привычных для большинства инженеров-разработчиков, ввода дополнительных данных непосредственно во время вычислительного процесса и др. Подобные возможности могут предоставить новый опыт для пользователей графоориентированного подхода [18], поскольку в таком случае графовая модель становится не только инструментом для формального описания алгоритмов, реализующих сложные вычислительные методы, но и инструментом описания логики работы пользователя в системе.

Чтобы не вносить существенных изменений в логику обхода графовой модели, реализованную в рамках библиотек pycomsdk и comsdk, было решено задействовать стандартные потоки вывода stdout и stderr для передачи статуса обхода графа. Пусть во время обхода графовая модель записывает в stdout успешно пройденные состояния, таким образом их наличие можно однозначно интерпретировать, как текущий статус решения. Если вывод в stderr не пуст, то это можно интерпретировать как ошибку в результате обхода графовой модели и соответствующим образом обрабатывать. Просмотр информации из управляющего процесса осуществляется с помощью объекта исполнительного процесса, у которого существуют поля stdout и stderr.

Для того чтобы текущий статус обхода графовой модели на стороне веб-клиента отображался в реальном времени, весь процесс получения статуса должен быть итеративным. На рис. 5, 6 приведены схемы обработки запросов веб-клиента на получение текущего статуса обхода графовой модели. Для обновления статуса обхода веб-клиент с заданным периодом по времени генерирует запросы на сервер приложений и визуализирует получаемые ответы. Данные от обработчика удаленного запуска с сервера приложений передаются на веб-клиент, где визуализируются на построенной графовой модели посредством раскраски ребер различными цветами в зависимости от ситуации (см. далее). Если была обнаружена ошибка, то на веб-клиент отправляется сообщение, поясняющее причину ошибки.

Тестирование. Для проведения тестирования реализованных программных средств был удаленно развернут сервер приложений pycomaps и веб-клиент comwpc. Инструментарий разработчика для работы с графовыми моделями на языке C++ был также удаленно собран, чтобы иметь доступ к библиотекам функций-обработчиков и функций-предикатов. Тестирование производилось вручную. С помощью веб-клиента в системе была создана и далее вызвана функция GRPH_SOLVER_WEB, являющаяся функциональным компонентом, относящимся к классу PYWEBS [1], что, в свою очередь, предписывает применение предварительно разработанных программных средств для автоматической генерации графического пользовательского интерфейса ввода данных на основе файла в формате aINI (листинг 1, файл grph_solver_web.slw) [28]. На рис. 7 представлен результат генерации.

Листинг 1

```
[MAIN] //Основные настройки решателя
SOLVER_SID=[[gcddot/gcdhom/test.adot]]
//Идентификатор решателя
IN_FNAME=[epoxy_xt118_a10_b3_GEOA03220.tsk] //Файл
входных данных
-FINISH_NODE=[HOM_STRENGTH_CALCULATED]$com.mstts
//Конечное состояние
DebugMode=[1]{0|1} //Режим отладки
PAUSE_NODE=[HOM_STRENGTH_CALCULATED]$com.mstts
//Узел для приостановки расчета
```

Для прикладных задач связь конкретного графоориентированного решателя с конкретной функцией реализуется с помощью определения соответствующих атрибутов функции в системной базе данных (настройки могут быть найдены в таблице sys.aiprv). Однако для нужд тестирования и бóльшей наглядности в представленной форме ввода был добавлен атрибут SOLVER_SID (листинг 1), обеспечивающий непосредственное

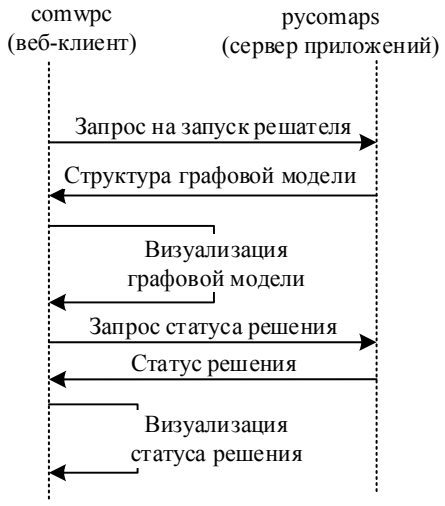


Рис. 5

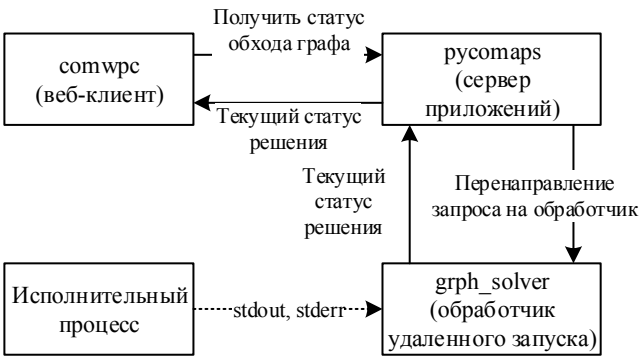


Рис. 6

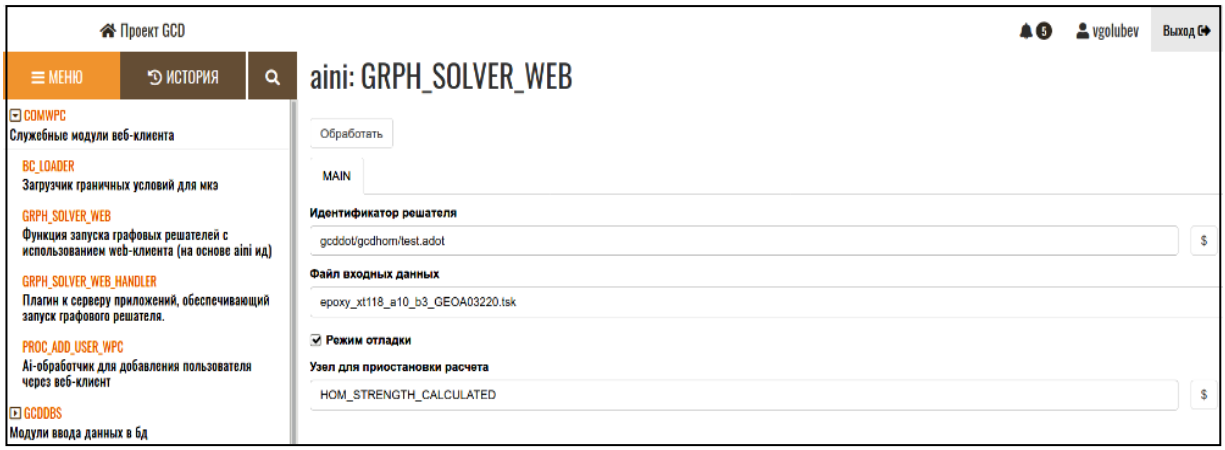


Рис. 7

определения имени aDOT файла тестовой графовой модели (листинг 2, файл test.adot). Напомним, что размещение файлов, определяющих различные графовые модели на языке aDOT, локализовано в специальном git-репозитории [1]. Представленный файл (листинг 2) описывает простой граф из трех узлов. К ребрам графа привязаны тестовые функции-обработчики и функции-предикаты из динамической C++ библиотеки libcomsdk.

Листинг 2

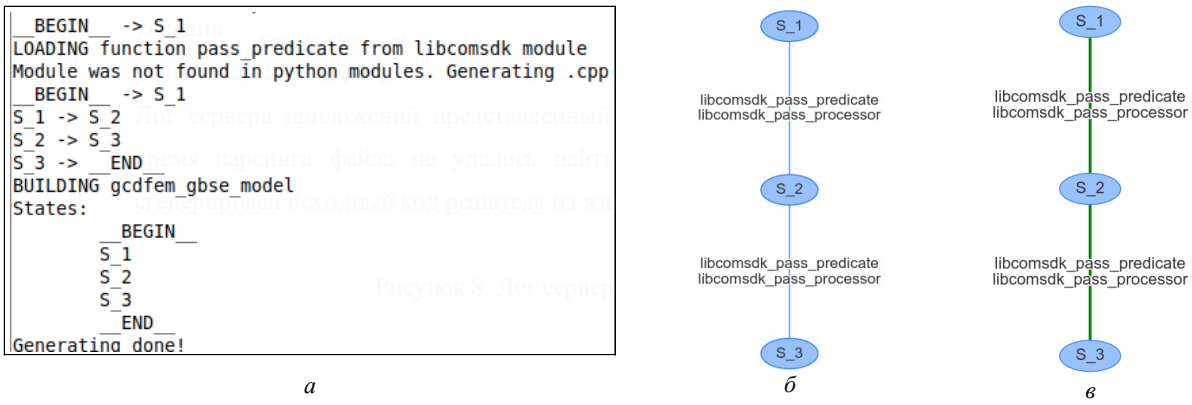
```
digraph gcdfem_gbse_model
{
// Определение функций-обработчиков
  PASS_PROCESSOR [module=libcomsdk, entry_func=pass_processor]
// Определение функций-предикатов
  PASS_PREDICATE [module=libcomsdk, entry_func=pass_predicate]
// Определение морфизмов
  PASS_MORPHISM [predicate=PASS_PREDICATE, function=PASS_PROCESSOR]
// Определение топологии графовой модели метода конечных элементов
  __BEGIN__ -> S_1
  S_1 -> S_2 [morphism=PASS_MORPHISM, comment = "Препроцессинг завершен, осуществляем подготовку к расчету..."]
  S_2 -> S_3 [morphism=PASS_MORPHISM, comment = "Расчет завершен, осуществляем подготовку к постпроцессингу..."]
  S_3 -> __END__ [comment = "Постпроцессинг завершен. Расчет завершен."]
}
```

Динамически были загружены скрипты, отвечающие за обработку браузерных событий, вызванных пользователем.

Для старта процесса выполнения необходимо нажать кнопку «Обработать» (рис. 7), что сопровождается: а) формированием и отправкой запроса на сервер приложений о необходимости запуска определенного решателя; б) сериализацией, отправкой, приемом и десериализацией параметров и их значений, определенных пользователем в форме ввода (рис. 7), на удаленный сервер согласно алгоритму, представленному на рис. 1, *б*, с последующим формированием объекта «*данных сложного вычислительного метода*»; в) вызовом специального функционального компонента, обеспечивающего обработку запроса и удаленный запуск решателя (функция GRPH_SOLVER_WEB_HANDLER), что осуществляется чтением и интерпретацией связанного файла в формате aDOT (листинг 2, файл test.adot), созданием объектов «*графовая модель*» и сопоставлением начального состояния созданной графовой модели и созданного объекта «*данные сложного вычислительного метода*».

Требовалось визуализировать исполняемую графовую модель и динамически отображать статус выполнения функций-обработчиков, связанных с ребрами, подсвечивая их тем или иным цветом: зеленым – в случае успеха (предикат отработал успешно, обработчик также успешно); красным – в случае неудачи (предикат запретил выполнение обработчика); в случае, если предикат разрешил выполнение обработчика, а обработчик завершился с ненулевым кодом ошибки, – желтым.

Структура графовой модели отправлялась на веб-клиент в формате JSON, далее интерпретировалась и визуализировалась в виде ориентированного графа с множеством состояний и функций перехода, которые соответствуют заданному графоориентированному решателю (рис. 8, *б*, *в*). На рис. 8, *а* представлен снимок экрана консоли вывода сообщений, визуализирующей процесс интерпретации графовой модели при ее обходе. Из представленных данных следует, что во время интерпретации не удалось найти python-модуль



а *б* *в*

Рис. 8

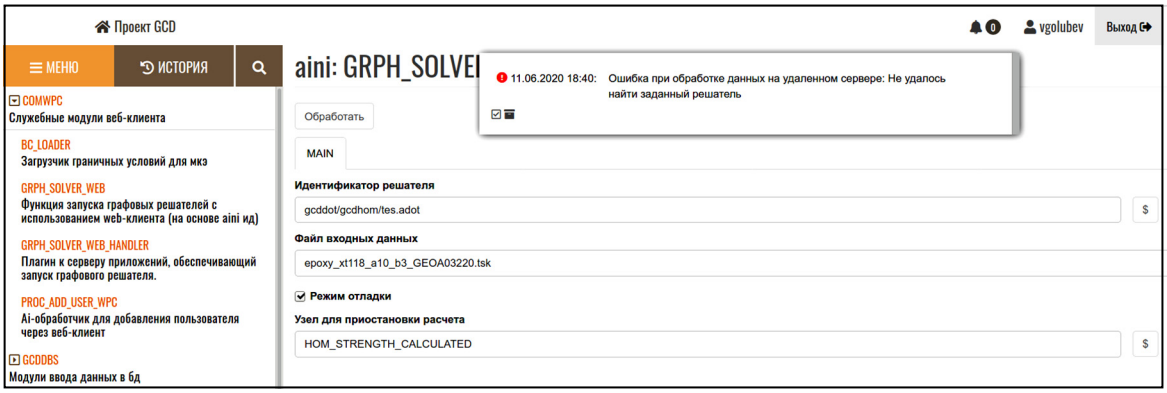


Рис. 9

libcomsdk, поэтому построенная графовая модель описывается исходным кодом на языке C++, который был автоматически сгенерирован с помощью инструментов pycomsdk. На рис. 8, *б* и *в* представлены снимки экранов, отражающие работу разработанной программы визуализации процесса обхода графовой модели: *б* – до обхода, *в* – после успешного обхода.

Одновременно проводилось тестирование возможностей отслеживания возникновения исключительных ситуаций и их обработки с помощью разработанной системы уведомлений. Для тестирования сценария неверного задания решателя был указан несуществующий в хранилище графовых моделей файл aDOT. В результате обработки функциональный GRPH_SOLVER_WEB_HANDLER не обнаружил файл, сформировал и отправил сообщение об ошибке, веб-клиент обнаружил в ответе сервера сообщение о возникшей ошибке и зарегистрировал событие типа «ошибка», связанное с сессией текущего пользователя, сформировал сообщение, пригодное для визуализации в пользовательском интерфейсе, и показал пользователю всплывающее окно с информацией о возникшем событии (рис. 9).

В случае возникновения ошибок процесс выполнения должен быть корректно завершен, в панели уведомлений должен быть инкрементирован счетчик и в списке сообщений должно быть доступно описание возникшей ситуации. Панель уведомлений применяется для отображения сообщений разных типов: информационных, предупреждений и ошибок. Тип стандартного сообщения хранится вместе с самим сообщением в системной базе данных (таблица sys.messg). В случае обработки исключительной ситуации могут применяться как стандартные, так и пользовательские сообщения.

При параллельном режиме обхода графовой модели и при использовании графовой модели для описания логики работы пользователя представленный метод визуализации, с точки зрения авторов, позволит наглядно демонстрировать местоположение возникающих проблем в процессе решения сложной вычислительной задачи, включая задачи автоматизированного проектирования сложных технических объектов.

Практический пример. На рис. 10 представлена перспективная графовая модель верхнего уровня абстракции, реализующая методологию автоматизированного проектирования КМ в рамках разрабатываемой САПР КМ, построение которой основано на представленном в настоящей статье методе организации вычислительных процессов.

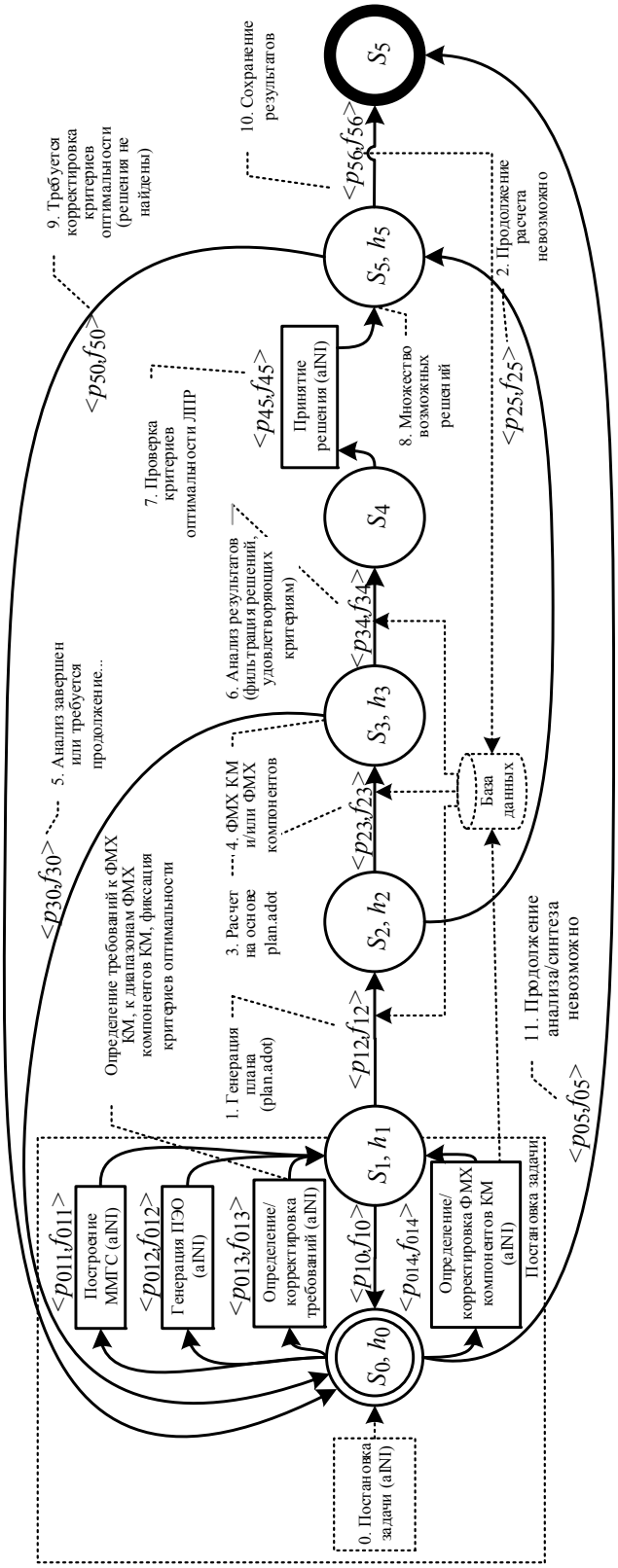

0. Постановка задачи (alNI)
Постановка задачи
Построение ММГС (alNI)
Генерация ПЭО (alNI)
Определение/ корректировка требований (alNI)
Определение/ корректировка ФМХ компонентов КМ (alNI)
Принятие решения (alNI)
База данных
S_0, h_0
S_1, h_1
S_2, h_2
S_3, h_3
S_4
S_5, h_5
S_5
$<p_{05}, f_{05}>$
$<p_{10}, f_{10}>$
$<p_{011}, f_{011}>$
$<p_{012}, f_{012}>$
$<p_{013}, f_{013}>$
$<p_{014}, f_{014}>$
$<p_{12}, f_{12}>$
$<p_{23}, f_{23}>$
$<p_{34}, f_{34}>$
$<p_{45}, f_{45}>$
$<p_{56}, f_{56}>$
$<p_{50}, f_{50}>$
$<p_{25}, f_{25}>$
$<p_{30}, f_{30}>$
1. Генерация плана (plan.adot)
3. Расчет на основе plan.adot
4. ФМХ КМ и/или ФМХ компонентов
5. Анализ завершен или требуется продолжение...
6. Анализ результатов (фильтрация решений, удовлетворяющих критериям)
7. Проверка критериев оптимальности ЛПР
8. Множество возможных решений
9. Требуется корректировка критериев оптимальности (решения не найдены)
10. Сохранение результатов
2. Продолжение расчета невозможно
11. Продолжение анализа/синтеза невозможно
Определение требований к ФМХ КМ, к диапазонам ФМХ компонентов КМ, фиксация критериев оптимальности

Рис. 10

Прямоугольниками выделены проектные процедуры, которые требуют участия лица, принимающего решение.

Пользователь формирует исходное описание проектируемого КМ в форме его многомасштабной многокомпонентной гетерогенной структуры (ММГС) [30], допускающей интервальные оценки физико-механических характеристик (ФМХ) отдельных компонентов. Осуществляется построение геометрических моделей представительных элементов объема (ПЭО), описывающих микроструктуру армирования проектируемого КМ и его гетерогенных компонентов на разных масштабных уровнях. Определяются известные ФМХ компонентов КМ либо их интервальные оценки. В качестве проектных требований должны быть определены ограничения на эффективные ФМХ проектируемого КМ, что также, как правило, возможно в форме их интервальных оценок. Далее следует автоматическое формирование вложенной графовой модели (рис. 10, п. 1), динамически определяющей план решения поставленной задачи с применением метода реверсивной многомасштабной гомогенизации (РМГ) [31], основанного на итеративной постановке и решении прямых и обратных задач микромеханики КМ с использованием определенных входных данных и базы данных ФМХ различных материалов, также разрабатываемой в рамках САПР КМ. Анализ результатов осуществляется в автоматизированном режиме: система должна предоставить пользователю массив возможных вариантов для выбора.

Обычно КМ различны как по структуре, так и по типам применяемых компонентов. Для задачи проектирования КМ с заданными ФМХ характерно отсутствие достаточной информации о ФМХ компонентов. Вычислительный метод исследования ФМХ КМ зависит от того, какие свойства и каких компонентов неизвестны. Процедура расчета ФМХ КМ состоит из конкретных подэтапов: а) определение компонентов с неизвестными свойствами; б) определение вспомогательных КМ с известными, например из эксперимента, ФМХ; в) рекурсивное определение и решение вспомогательных коэффициентных обратных задач идентификации неизвестных параметров; г) в случае успешной идентификации параметров постановка и решение задачи многомасштабной гомогенизации для исходного КМ. Таким образом, процедура расчета ФМХ КМ с неопределенными параметрами сводится к многократной постановке и решению прямых и обратных задач гомогенизации, что осуществляется с помощью конкретных «решателей», реализуемых конкретными графовыми моделями. В зависимости от типа исследуемых ФМХ в данных графовых моделях будут лишь заменяться отдельные функции-обработчики. Детальное изложение материала выходит за рамки настоящей статьи.

Разработка программной реализации указанной методологии автоматизированного проектирования КМ также ведется в настоящее время.

Важной задачей, которая была решена в части доработки ранее созданных программных инструментов, реализующих графоориентированный подход, стало обеспечение возможностей приема, передачи и интерпретации сообщений к графовой модели.

Полученные результаты создают основу для разработки новых графоориентированных модулей расширения функциональных возможностей САПР КМ, обладающих свойством интерактивного взаимодействия пользователя с вычислительным процессом, включая возможности организации вычислительных экспериментов с помощью графического пользовательского интерфейса.

Созданные программные решения позволяют конструировать сложную наукоемкую систему, в том числе определять бизнес-логику работы пользователей с различными ролями, скрывая рутинные операции и снижая потребность в документации.

Представленная методология приводит к существенному упрощению работы пользователя в системе за счет того, что система «сама сопровождает» пользователя на каждом этапе сложного вычислительного процесса, запрашивая дополнительные данные по мере их необходимости. В результате это приводит к снижению требований к квалификации пользователя.

Представленная методология позволяет автоматизировать процесс построения «полезной для пользователя» документации о методах решения конкретных прикладных задач. За счет наличия связанной графовой модели, определяющей логику работы пользователя в системе, появляется возможность генерации актуальной документации посредством холостого обхода модели с синхронным анализом массива формируемых данных, что, в свою очередь, возможно за счет их хранения в едином формате [1]. Более того, обеспечивается требуемое и естественное снижение

потребности в актуальной документации (см. утверждение 1), так как связанные графовые модели модифицируются разработчиком САПР, а их обход не требует от пользователя специальных усилий. Первая версия программной реализации такой методики построения документации была осуществлена и представлена в [2], при этом использовались шаблоны, построенные на языке LaTeX. Создание генератора документации, анализирующего идентификаторы обрабатываемых данных при обходе связанной модели, представляет собой предмет будущих работ.

Полноценная программная реализация методологии, ее тестирование и отладка также является предметом будущих работ. Цель разработки состоит в создании универсальной программной инфраструктуры разработки наукоемкого ПО и, в частности, САПР в различных предметных областях, включая возможности организации распределенных вычислительных процессов.

Разработка обеспечила применение принципов реактивного и потоко-ориентированного программирования при создании новых модулей в рамках представляемого САПР КМ. В ее основе лежат разработанные доцентом А. П. Соколовым и ассистентом А. Ю. Першиным базовые принципы и методики графоориентированной разработки программного обеспечения, реализованные в форме библиотек функций, форматов и структур данных.

Представленные научно-технические решения разработаны авторами в МГТУ им. Н. Э. Баумана на инициативной основе.

СПИСОК ЛИТЕРАТУРЫ

1. Соколов А. П., Першин А. Ю. Система автоматизированного проектирования композиционных материалов. Ч. 2. Вычислительная подсистема, распределенные вычисления с применением графоориентированного подхода // Изв. СПбГЭТУ «ЛЭТИ». 2020. № 10. С. 49–63.

2. Разработка программного обеспечения генерации кода на основе шаблонов при создании систем инженерного анализа / А. П. Соколов, В. М. Макаренков, А. Ю. Першин, И. С. Лаишевский // Программная инженерия. 2019. Т. 10, № 9–10, С. 400–416.

3. Норенков И. П. Основы автоматизированного проектирования. М.: Изд-во МГТУ им. Н. Э. Баумана, 2006.

4. Соколов А. П., Першин А. Ю. Система автоматизированного проектирования композиционных материалов. Ч. 1. Концепции, архитектура и платформа разработки // Изв. СПбГЭТУ «ЛЭТИ». 2020. № 8–9. С. 72–83.

5. McCormic B. H, DeFanti T. A., Brown M. D. Visualization in scientific computing. New York: Computer Graphics. 1987.

6. McIlroy M. Mass produced software components // Proc. of NATO Software Engineering Conf. (7–11 Oct. 1969), Garmisch, Germany: Scientific Affairs Division. 1969. P. 138–158.

7. ISO/IEC 19501:2005. Information technology – Open distributed processing – Unified Modeling Language (UML). Version 1.4.2. ISO.org. 2005. URL: https://www.iso.org/standard/32620.html (дата обращения 18.12.20).

8. Pat. US 2004/0056908 Al. Method and system for dataflow creation and execution / R. D. Bjornson et al. Assignee: Turbo Worx, Inc. (New Haven, CT, US). Publication date: Mar. 25, 2004.

9. Pat. US 8,949,772 Bl. Dynamic model-based software application development / D. TalbyScott, D. McMaster. Assignee: Amazon Technologies, Inc. (Reno, NV, US). Publication date: Feb. 3, 2015.

10. Леонтьев Д. В., Тарасов Г. В., Харитонов И. Д. Модель обработки данных вычислительных экспериментов // Научный сервис в сети Интернет: тр. XX Всерос. науч. конф. Москва: ИПМ им. М. В. Келдыша. М., 2018. С. 373–386.

11. Zurek R. W, Martin L. J. GridPP: development of the UK computing grid for particle physics // J. of Physics G: Nuclear and Particle Physics. 2006. Vol. 32. P. 1–20.

12. Алекперов Р. К. Организация распределенных вычислений на базе GRID-технологии // Искусственный интеллект. 2011. № 1. С. 6–14.

13. Farkas Z., Kacsuk P. P-GRADE portal: a generic workflow system to support user communities // Future Generation Computer Systems. 2010. Vol. 27, № 5. P. 454–465.

14. Кудрявцев А. О., Кошелев В. К., Избышев А. О. Разработка и реализация облачной системы для решения высокопроизводительных задач // Тр. Ин-та системного программирования РАН. 2013. Т. 24. С. 13–33.

15. Orange: Data Mining Toolbox in Python / J. Demsar, T. Curk, A. Erjavec et al. // J. of Machine Learning Research. 2013. № 14. P. 2349–2353.

16. Официальный сайт проекта TensorFlow. URL: https://www.tensorflow.org (дата обращения 05.01.2021).

17. Познаем Нирвану – универсальную вычислительную платформу Яндекса // База знаний Хабр. URL: https://habr.com/ru/company/yandex/blog/351016/ (дата обращения 05.01.2021).

18. Соколов А. П., Першин А. Ю. Графоориентированный программный каркас для реализации сложных вычислительных методов // Программирование. 2019. Т. 47, № 5. С. 43–55.

19. Apache Thrift. URL: https://thrift.apache.org/docs/ (дата обращения 06.01.2021).

20. Finagle Official Page. URL: https://twitter.github.io/finagle/ (дата обращения 06.01.2021).

21. gRPC Documentation. URL: https://www.grpc.io/docs/ (дата обращения 06.01.2021).

22. Wilde E., Pautasso C. REST: From research to practice. New York: Springer, 2011.

23. HDF5 Software Documentation. URL: https://support.hdfgroup.org/HDF5/doc/ (дата обращения 06.01.2021).

24. Белоушко К. Е. Формат NetCDF как стандарт для обмена данными в атмосферных исследованиях // Тез. II конф. «Базы данных, инструменты и информационные основы полярных геофизических исследований» (POLAR-2012). Троицк, ИЗМИРАН, 2012.

25. Tanenbaum A. S. Distributed systems: principles and paradigms. Upper Saddle River, NJ: Pearson Prentice Hall. 2006. URL: https:// www.studmed.ru/view/tanenbaum-e-raspredelennye-sistemy-principy-i-paradigmy_6f5579549ea.html (дата обращения 05.01.2021).

26. Аветисян А. И., Грушин Д. А., Рыжов А. Г. Системы управления кластерами // Тр. Ин-та системного программирования РАН. 2002. Т. 3. С. 39–62.

27. Пат. RU 2681408. Способ и система графоориентированного создания масштабируемых и сопровождаемых программных реализаций сложных вычислительных методов / А. П. Соколов, А. Ю. Першин. Опубл. 22.02.2019.

28. Соколов А. П., Першин А. Ю. Программный инструментарий для создания подсистем ввода данных при разработке систем инженерного анализа // Программная инженерия. 2017. Т. 8, № 12, С. 543–555.

29. Соколов А. П., Першин А. Ю. Описание формата данных aDOT (advanced DOT). Облачный сервис SA2 Systems. URL: https://sa2systems.ru/nextcloud/index.php/f/403526 (дата обращения 09.01.2021).

30. Гомогенизация многоуровневых многокомпонентных гетерогенных структур для определения физико-механических характеристик композиционных материалов / А. П. Соколов, А. Ю. Першин, А. В. Козов, Н. Д. Кириллов // Физическая мезомеханика. 2018. Т. 21, № 5. С. 90–107.

31. Sokolov A. P., Pershin A. Yu. Computer-aided design of composite materials using reversible multiscale homogenization and graph-based software engineering // Key Engineering Materials. 2018. Vol. 779. P. 11–18.

A. P. Sokolov, V. O. Golubev
Bauman Moscow Technical University

THE SYSTEM OF COMPUTER-AIDED DESIGN OF COMPOSITE MATERIALS PART 3. GRAPH-BASED METHODOLOGY FOR DEVELOPING USER-SYSTEM INTERACTION TOOLS

Despite the active development of computer-aided software, it is obvious that applied e-science software, including CAE and CAD systems, cloud computing platforms continue to be available only to a limited number of specialists, which is due to their high cost and, as a rule, complexity in particular application. At the same time the need of such systems is only increasing for scientists, engineers, students. The current paper presents an original graph-based method for organizing computing processes in such systems, as well as a methodology based on it for developing specialized extension modules for the presented CAD system. This methodology allow to describe the logic of the user's work in the system to solve a specific application problem (computational problem, design new composite material, etc.). The analysis of existing software solutions in the field of organization of complex computing processes in a distributed computing environment was carried out and a list of important software tools that are necessary for the organization of the corresponding software infrastructure was presented. The relevance of the work in the chosen direction is justified. The method and methodology presented in this paper are designed to simplify both the processes of developing new features and the processes of working in complex engineering software. Description of some details of the implementation of the developed methodology are presented. Some results of testing the developed software tools that provide user interaction with the system in real time of the process of traversing the test graph model are presented. The created software tools allow you to visualize the current status of the graph model traversal in run-time. An example of a top level graph model that implements the methodology of computer-aided composite material design was given within the framework of the developed CAD system, which was built using the presented methodology.

Technologies for developing of engineering software, graph-based software engineering, business-logic description, reactive developing, low-code development, development of computer-aided engineering systems, extension modules development, distributed computing technologies, workflow systems, e-science tools, computer-aided design of composite materials