Новая программная архитектура для специализированных систем математических расчетов[1]

П.В. Куракин

Аннотация. Описана разработанная автором в ряде государственных контрактных работ оригинальная архитектура специализированных систем математических расчетов. Архитектура опирается на свободно распространяемые платформы, протоколы и пакеты прикладных программ: Java, JavaScript, HTTP, Python, Octave. Целевая область применения этой архитектуры: задачи разработки систем поддержки принятия решений (СППР); также приложения в области систем автоматизированного проектирования (САПР).

Ключевые слова: Java, JavaScript, HTTP, JSON, Python, Octave, математические расчеты, СППР, САПР.

Введение

Опыт работы автора в ряде государственных контрактных работ с предприятиями космической отрасли [1] показал, что в настоящее время все чаще оказываются востребованными специализированные системы математических расчетов, которые включают:

1. хранилища (расширяемые) данных всевозможных типов и форматов, связанных с соответствующей предметной областью;
2. библиотеки (расширяемые) вычислительных методов и математических моделей, характерных для соответствующей предметной области;
3. графический пользовательский интерфейс, представляющий собой либо включающий в себя как ключевой компонент визуальные средства создания и редактирования типовых расчетных задач, характерных для соответствующей предметной области;
4. вычислительная подсистема, позволяющая автоматически выполнять требуемые расчеты при наличии нужных входных данных;
5. программный механизм передачи данных (структурированного описания задачи, требующей тех или иных расчетов) от пользовательского интерфейса к вычислительной подсистеме (перед выполнением расчетов) и в обратном направлении (после выполнения расчетов).

Кратко и на качественном уровне проблему можно описать так: требуются программные средства, аналогичные популярной системе MATLAB [2] в совокупности с подсистемой графического редактирования задач Simulink [3], но в сочетании с некоторым хранилищем данных и описаний задач, при этом основанные на бесплатном (свободно распространяемом) программном обеспечении.

Ключевой недостаток стандартной связки пакетов MATLAB + Simulink (помимо того, что

[1] Работа выполнена при поддержке РФФИ грант №13-01-00617-а «Исследование новых типов и механизмов самоорганизации. Разработка алгоритмов анализа и прогноза динамики нелинейных систем».

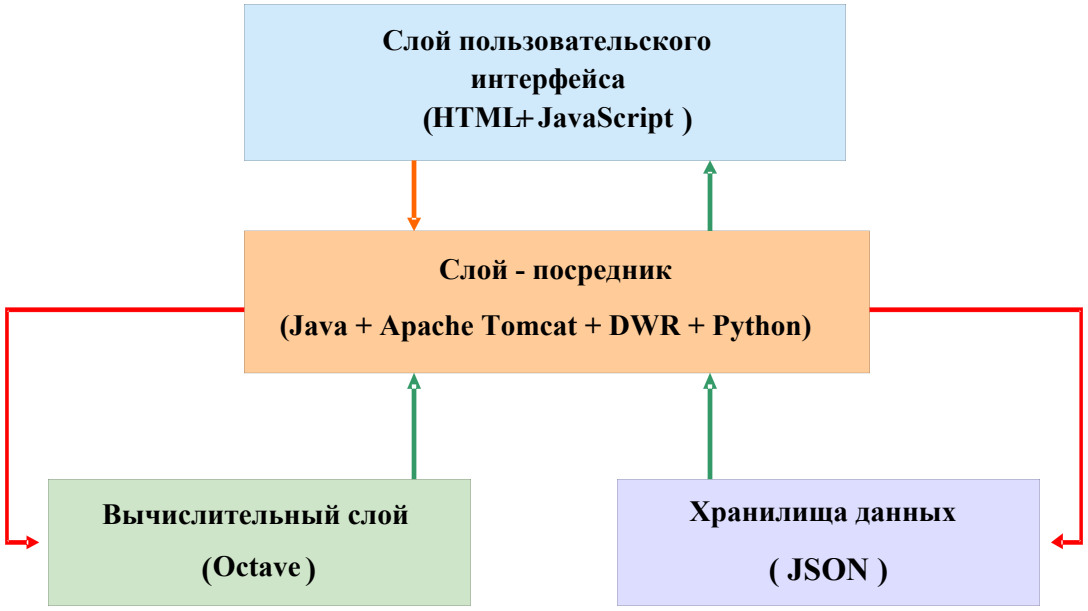

Рис. 1. Архитектура программной среды

это весьма дорогостоящий коммерческий продукт, т.е. практически непригоден для изготовления законных «коробочных» программных решений для государственных заказчиков) следующий: библиотека графических примитивов подсистемы Simulink, при ее кажущемся обилии, на деле ограничена некоторыми популярными и типичными инженерно-расчетными задачами.

Все большему количеству организаций нужен доступный программный инструмент математических расчетов в самых разнообразных прикладных областях с развитым инструментарием визуального редактирования задач, причем эти задачи, как правило, требуют специфических графических примитивов и способов их объединения в целостные графические описания этих задач.

Далее в тесте, для удобства, наряду с термином «архитектура» используются термины «решение», «архитектурное решение» и «программная среда», поскольку описываемая архитектура и есть решение, объединяющее в единую схему и единую среду ряд программных модулей, как готовых (присутствующих на рынке свободного ПО), так и разработанных в ходе выполненных исследовательских работ.

1. Общее описание архитектуры

Описываемая программная среда в целом укладывается в концепцию «клиент – сервер» [4] и опирается на платформу Java и веб – технологии (Рис. 1).

Выбор веб – технологий как технологической основы описываемой программной среды основан на следующих тезисах.

1. Решение должно опираться только на свободно распространяемое программное обеспечение и готовые библиотеки программного кода.

2. Опора на готовые решения свободного доступа означает, что пользовательский интерфейс и вычислительная подсистема среды будут реализованы как самостоятельные процессы операционный системы (ОС) компьютера; так или иначе, наиболее естественным и технологичным способом передачи данных между процессами является стек сетевых протоколов TCP \ IP; веб – технологии являются естественной надстройкой над стеком TCP \ IP.

3. Реализация графического интерфейса в среде веб - браузера (средствами стандартного браузерного языка программирования JavaScript) оказалась наиболее естественным

решением в силу предыдущих пунктов; имеются хорошо зарекомендовавшие себя библиотеки разработки графических пользовательских интерфейсов на языке программирования JavaScript.

4. С учетом пунктов 2 и 3 был сделан выбор: все решение следует реализовать как веб - приложение.

5. Решение создавать веб - приложение предопределяет и выбор Java как базовой платформы всей платформы, поскольку наиболее популярные, опробованные, а тем более бесплатные серверы веб - приложений (семейство Apache Tomcat) реализованы именно на Java.

С точки зрения количества «уровней» данную архитектуру можно с равным успехом характеризовать как трехуровневую, так и четырехуровневую. В некоторых системах классификации подсистемы «логики» и «данных» разносят на разные уровни. Автор предпочитает рассматривать эти подсистемы как находящиеся на одном уровне, тогда обо всей архитектуре уместно говорить как о трехуровневой.

Ключевую роль для функционирования описанной архитектуры играет язык разметки данных JSON [5], который исторически возник как раз для браузерных задач (аббревиатура расшифровывается как JavaScript Object Notation, иногда этот язык называют «обезжиренный XML»). JSON – документ представляет собой объект типа *словарь*, т.е. перечень вхождений вида <ключ> : <значение>, через запятую. Значение может быть как атомарным типом (включая массив), либо представлять собой объект (перечень включений). Каждый объект – перечень заключается в фигурные скобки.

Вся схема может работать, только если существует возможность передавать данные из одного процесса операционной системы в другой, желательно в виде текстовой строки, формат которой легко поддается расшифровке в обоих процессах. Стандарт JSON, как показал опыт разработки, лучше всего соответствует этой цели.

Как отмечено выше, самый естественный и полновесный способ связи различных процессов в любой операционной системе – стек сетевых протоколов TCP\IP. Но гораздо удобнее пользоваться этим стеком не непосредственно, а при помощи какой-то «настройки» - т.е., пользоваться протоколом уровня приложения, а не транспортного уровня [6]. Естественным ответом является протокол гипертекстовой передачи гипертекста HTTP [7] (термин гипертекст означает обычный текст в совокупности с управляющими символами разметки этого текста).

В целом, функционирование всей среды как системы математических расчетов основано на том, что протокол HTTP передает описание задачи в виде JSON – строки от пользовательского интерфейса (реализованного как браузерный графический редактор) к вычисляющему слою. После выполнения расчетов, данные снова преобразуются в JSON – строку, которая сохраняется в хранилище системы и может быть потом снова загружена в пользовательский интерфейс. Далее это процесс рассмотрен более детально.

2. Схема работы архитектуры

Интерфейс пользователя, как уже указано, размещается на веб – странице, поэтому изначально речь идет о стандартном веб – приложении с HTML – файлами, установленном на веб – сервере Apache Tomcat [8] (версия 6.0.33, работает на платформе Java, которая должна быть предустановленна на компьютере).

На стороне веб – браузера данные объектов JavaScript, созданные пользователем в графическом редакторе, конвертируются в строку в формате JSON (*сериализация* данных [9]). Это преобразование выполняется средствами бесплатной JavaScript – библиотеки Yahoo UI [10] (эта же библиотека использована для создания графического интерфейса пользователя). Эта строка посредством технологии DWR (см. далее) передается в вычислительную подсистему.

Архитектурное решение использует Java - библиотеку открытого кода DWR [11]. Из Рис. 1 видно, что библиотека DWR находится в центре описываемой архитектуры. Технология DWR (Direct Web Remoting) встраивает в исходное веб – приложение специальный *сервлет*. Обычный сервлет [12] представляет собой небольшой и относительно простой Java – класс, устанавливаемый (вместе с дополнительными служебными файлами) на веб – сервере. Класс сервлета умеет обрабатывать ко-

манды (запросы) протокола HTTP, поступающие к серверу со стороны веб - страницы. В случае технологии DWR класс сервлета генерируется *автоматически*.

Технология DWR работает так. На стороне веб – сервера пользователь этой технологии размещает (самостоятельно, в отличие от тех классов, которые автоматически создает DWR) один или несколько собственных Java – классов и регистрирует их в конфигурационных файлах DWR (на сервере), а также на веб – странице приложения при помощи специальной директивы языка разметки HTML. При старте приложения (при обращении к веб – странице по сети по протоколу HTTP) в пространстве оперативной памяти браузера создаются JavaScript объекты – дублеры серверных Java – классов пользователя. Эти объекты имеют те же методы, что серверные классы. С практической точки зрения достаточно иметь всего один такой Java – объект на сервере, назовем его *диспетчером*.

На стороне веб – страницы пользователь может вызывать JavaScript - методы объекта - дублера, что приводит к вызову одноименных методов серверного Java – объекта. Все такие методы серверного класса возвращают строку, и эта строка *асинхронно* возвращается (через сеть, т.е. средствами протокола HTTP) в браузер.

Таким образом, технология DWR позволяет создать на стороне веб – страницы «пульт удаленного управления» серверного Java – объекта (диспетчера) и получать от него результирующие данные в форме текстовой строки. Вызываемые удаленно методы суть методы, обращающиеся к хранилищам данных, либо методы, вызывающие на исполнение вычислительную подсистему. Разумеется, при вызове вычислительной подсистемы диспетчер передает ей входные данные, полученные в виде JSON – строки от браузера.

В качестве вычислительной подсистемы используется пакет прикладных математических расчетов Octave [13], который номинируется как функционально полный и бесплатный аналог пакета MATLAB. На стороне пакета Octave используется еще одна библиотека открытого кода для конвертирования JSON – строки в объекты языка программирования Octave (*десериализация* данных), который совпадает с языком программирования MATLAB. Это позволяет обрабатывать полученные данные средствами языка программирования Octave\MATLAB. После проведения вычислений объекты языка программирования Octave снова конвертируются в JSON – строку (снова сериализация), которая сохраняется в файл. Такие файлы мы называем *конфигурациями*. Этот JSON – файл снова можно загрузить (по HTTP – запросу через указанное выше сервлетное приложение) в среду графического интерфейса в браузере.

Необходимо отметить, что среда Octave запускается на исполнение асинхронно; решение записывается в исходный файл JSON – конфигурации задачи. При этом, исходный файл переписывается; поддерживается идеология одна задача – один файл; обработка конфигурации означает изменение значений тех или иных полей либо появление определенных значений в тех полях, где были указаны неопределенные значения (“null” или “undefined”). Автоматической отправки решения клиенту не происходит; для загрузки решения в браузер требуется отправка отдельного запроса. В этом смысле решенная задача вообще никак не выделена и не отличается от решенной – с точки зрения системы и то и другое есть один конфигурационный файл в формате JSON.

Для работы всей системы необходимо разместить где-то стартовые настройки всего приложения (где находятся хранилища данных, где находятся математические алгоритмы, где находится Python - код). Текстовый файл с такими настройками уместно расположить, например, в главном каталоге /bin веб - сервера.

3. Хранилища данных

Важно подчеркнуть, что формат JSON используется не только для передачи данных, описывающих конфигурацию решаемой задачи, но и для хранения всех сопутствующих данных. Из Рис. 1 видно, что нижний слой трехзвенной архитектуры программной среды состоит не только из вычислительной подсистемы, но из хранилища данных (текстовые файлы в формате JSON). Эти данные представляют собой совокупность как конфигураций

задач, так и всевозможных сопутствующих технических данных из соответствующей предметной области, которые используются при моделировании. Например, когда описываемая архитектура применялась в задачах оценочного моделирования пилотируемой экспедиции на Луну, создавались и использовались хранилища данных по космическим средствам, бортовым системам, полетным операциям и т.п.

Необходимо подчеркнуть, что практика разработки и эксплуатации программной среды показала, что использование традиционных СУБД, основанных на реляционной модели данных, здесь нецелесообразно. Реляционная модель представляется слишком жесткой для тех задач, для которых создавалась архитектура. Говоря кратко, в виде таблиц удобно представлять только очень ограниченные по типу массивы информации. В основном, эта модель данных подходит только для задач учета (продукции, персонала и т.п.).

Стандарт JSON (как «обезжиренный XML») соответствует древовидной (иерархической) организации данных. Практически, это соответствует понятию «объекта» в объектно-ориентированном программировании (ООП), что чаще всего и есть лучшая абстракция данных в задачах математического моделирования сложных систем. Ниже приведен пример небольшого JSON – описания объекта «Посадочная ступень лунного модуля при 2-пусковой схеме»:

```
  {
  "description_": "Posadochnaya stupen' lunnogo korablya v 2-puskovoi skheme",
  "type": "Lander unit of Moon landing ship 2 start",
  "name": "LU_MLS2",
  "dry_mass": "5000.0",
  "fuel_mass": "10000.0",
  "exhaust_velocity": "3000",
  "bs_list": ["EngineSystem", "ComplementSystem"],
  }
```

Как можно видеть, этот объект состоит из описания, указания типа данного космического модуля, его названия, сухой массы, массы топлива, требуемой скорости истечения топлива и списка бортовых систем.

Хранилища можно организовывать произвольным образом. В принципе, это может быть просто один каталог в файловой системе, но разумнее сделать систему каталогов по тематическому признаку (отдельные каталоги).

4. Использование Python

Место языка программирования Python в описываемой программной среде следующее. Набор методов, которые (посредством технологии DWR) требуется удаленно вызывать на сервере, меняется в зависимости от конкретной реализации всей системы. Для этого приходится переписывать заново (и перекомпилировать) Java - класс диспетчера. Использование языка программирования Python позволило поступить более гибко; одновременно заложена основа для дальнейшего развития архитектуры.

Гибкость достигается следующим образом. Класс диспетчера написан так, чтобы его перекомпиляция больше не требовалось («раз и навсегда»): набор методов этого класса фиксирован. Но это не отменяет возможности для расширения набора методов, доступных для удаленного вызова, вообще. Фактически, класс диспетчера имплементирует (реализует) всего один метод, который передает управление подсистеме центрального слоя (согласно Рис.1), реализованной на языке программирования Python.

Эта подсистема состоит просто из набора текстовых файлов с программным кодом на языке Python. Имя конкретной требуемой процедуры передается в среду Python как строка; набор параметров этой процедуры передается как массив строк. Как было отмечено выше, вызовы Python – процедур функционально сводятся к вызову вычислительной подсистемы (Octave) либо работе с хранилищем вспомогательных данных и конфигураций задач (Рис 2). Библиотека используемых вычислительных алгоритмов (файлы имеют стандартное для MATLAB \ Octave расширение ”.m”) рассматривается как часть хранилища данных.

Благодаря такой организации цепочки удаленных вызовов удалось достичь высокой гибкости и адаптивности архитектуры к совершенно различным предметным областям. Python – код можно оперативно обновлять,

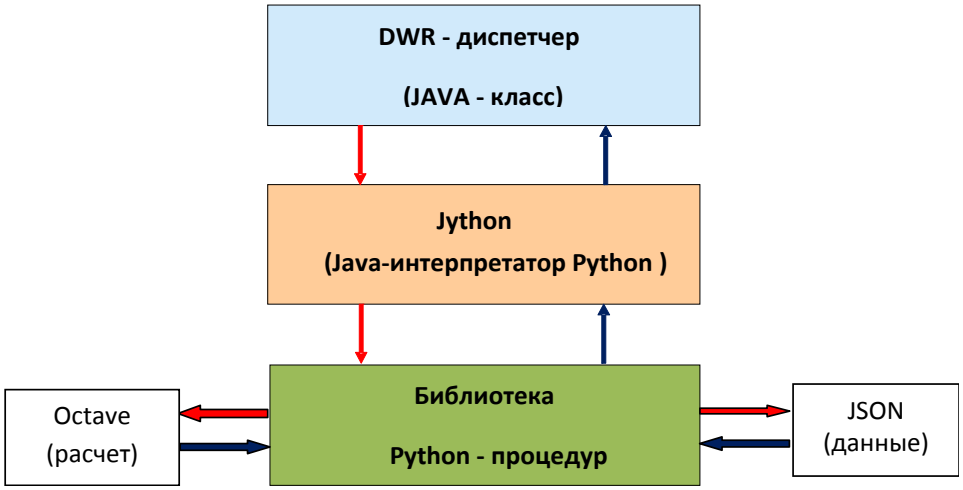

Рис. 2. Детализация среднего и нижнего слоев архитектуры

не затрагивая центральное ядро архитектуры (DWR – слой); компиляции этого кода не требуется – он вызывается на исполнение интерпретатором Jython.

Принципиально описанная цепочка вызовов представляет собой одну из возможных реализаций концепции удаленного вызова процедур RPC (Remote procedure call) [15]. Роль языка программирования Python в дальнейшем развитии архитектуры и обсуждение результатов описаны в разделе Заключение.

5. Интерфейс пользователя и графический редактор

Как отмечено во Введении, вся описываемая программная среда разрабатывалась, по сути, для того чтобы создать инструментарий, позволяющий специалисту создавать средствами *визуального проектирования* описания задач в его предметной области. Поэтому графический редактор – функционально самая важная часть архитектуры.

На Рис. 3 приведен пример графического представления типичной конфигурации в системе оценочного моделирования космический миссий. Конфигурация создается пользователем из некоторого набора графических примитивов. В данной реализации программной среды использовались всего два примитива – «событие» (или, что то же – «полетная точка») и «операция». Событию соответствует графический примитив «прямоугольник», операция создается как отрезок с кружочком, соединяющий два события. В другой реализации среды, созданной для моделирования производствен-

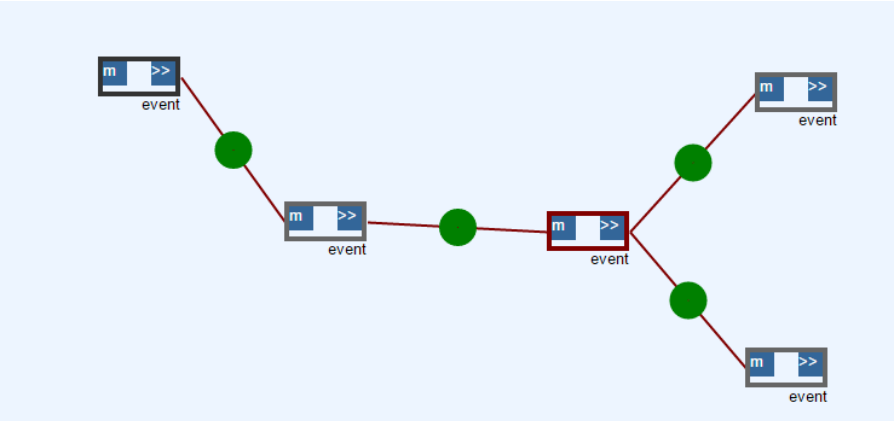

Рис. 3. Представление типовой конфигурации задачи в среде браузерного графического редактора

но-экономических задач, прямоугольникам соответствуют промышленные предприятия, а отрезкам – поставки продукции между предприятиями.

В обоих этих случаях формальное описание задачи представляет собой, в целом, математический граф, т.е. совокупность узлов и ребер. Узлы графа (прямоугольники) создаются по принципу “drag and drop” («захвати и урони»), как и в большинстве коммерческих средств визуального проектирования (например, упомянутый во Введении Simulink или Visio компании Microsoft). Чтобы создать ребро графа («операцию» или «поставку продукции») надо схватить мышью символ «>>» на графическом примитиве узла и перетащить его до следующего (целевого) узла – прямоугольника. Принципиально, узлы графа могут представлять объекты произвольной природы, а ребра графа – те или иные бинарные связи таких объектов.

Графические примитивы создаются при помощи открытой JavaScript – библиотеки Raphael [16]. В целом рынок открытых JavaScript – библиотек бурно развивается в последние годы, что также послужило, на начальном этапе разработки архитектуры, дополнительным стимулом построить искомую архитектуру специализированных систем математических расчетов на основе веб-технологий.

Главная особенность графических примитивов, создаваемых в среде графического редактора – к ним можно привязывать наборы данных и редактировать их (Рис. 4). Как можно видеть из рисунка, набор данных, привязанных к отдельному примитиву (соответственно – объекту, который этот примитив представляет) имеет древовидную структуру; допустима произвольная глубина вложенности данных, иными словами – узлам и ребрам графа задачи можно сопоставлять объекты произвольной сложности.

В совокупности с механизмом построения графа это составляет полный механизм создания и редактирования конфигураций задач. Как отмечено выше, созданную описанным способом конфигурацию можно средствами библиотеки Yahoo UI конвертировать в JSON – строку, а далее, на основе описанного выше механизма

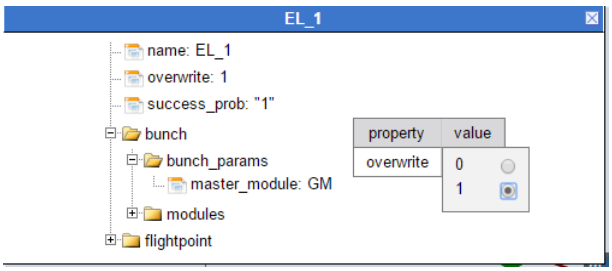

Рис. 4. Меню редактирования параметров объекта

удаленного вызова соответствующей процедуры (реализованной на языке Python), сохранить как текстовый файл в хранилище конфигураций. Аналогичным путем можно вызвать процедуру обработки ранее сохраненной конфигурации (расчет), а также загрузки ранее созданной конфигурации (неважно – обработанной или нет).

На данный момент реализован графический редактор, предполагающий совокупную структуру данных, описывающих задачу в виде графа. Принципиально технологии Raphael и Yahoo UI позволяют разработать средство визуального проектирования для любых других сложных структур данных.

6. Возможные альтернативные архитектурные решения

Естественно возникает вопрос – существуют ли альтернативные программно-архитектурные решения задач, перечисленных во Введении. Мнение автора таково: в любом случае наиболее продуктивный путь решения таких задач состоит в использовании некоторой комбинации готовых (бесплатных) платформ и библиотек. Так или иначе, в том или ином виде, для стыковки этих платформ потребуется реализовать идеологию удаленных вызовов процедур RPC [16]. В сочетании с требованием наличия развитого графического интерфейса это наиболее вероятно ведет к опоре на веб - технологии.

Отправной точкой проектирования, на самом деле, является графический редактор пользовательского интерфейса, при помощи которого рассматриваемый пользователь (это специалист – предметник из государственных и производственных структур) создает графиче-

ское представление (затем – и полное описание) своей задачи. Сочетание высокой выразительности и функциональности с требованием свободного распространения программного кода, лежащего в основе такого инструмента, практически неизбежно отправляет к существующим JavaScript – библиотекам. Это объясняется взрывным ростом пространства веб – приложений массового использования в последние два десятилетия.

Одновременно с выбором веб - среды разработки графического редактора пользовательского интерфейса становится естественной опора на платформу Java – одну из самых популярных и опробованных платформ для веб – приложений (данный язык программирования исторически и разрабатывался как язык серверного программирования для распределенных систем [18]).

Представляется, что разумные альтернативы предложенному решению могут представлять собой реализацию идеологии удаленного вызова процедур RPC [16]. Эти реализации могут использовать как веб – технологии как обертку над сетевыми протоколами, так и другие надстройки. Необходимость в развитом пользовательском интерфейсе с графическим редактором по-прежнему делает привлекательным использование готовых графических библиотек, реализованных на JavaScript. А это значит, что опора на веб остается наиболее ожидаемым направлением.

При этом уже используемый в описанном решении язык программирования Python представляет в настоящее время вторую по популярности после Java среду разработки всевозможных расширений. В том числе, уже существуют бесплатные веб – серверы, написанные на Python и работающие как интерпретируемые приложения при установленной платформе Python. В настоящее время эти серверы менее опробованны и надежны.

Заключение

Описанная архитектура отвечает функциональным требованиям, заявленным во Введении: «Кратко и на качественном уровне проблему можно описать так: требуются программные средства, аналогичные популярной системе MATLAB [2] в совокупности с подсистемой графического редактирования задач Simulink [3], но в сочетании с некоторым хранилищем данных и описаний задач, при этом основанные на бесплатном (свободно распространяемом) программном обеспечении».

В целом, такая программная среда подходит для средств поддержки принятия решений (СППР [17]), предполагающих математические расчеты на основе некоторой, потенциально произвольно сложной и достаточно большой по размеру, структуры данных из соответствующей предметной области. Конкретно, данная архитектура была применена для задач моделирования производственно-экономических задач в космической отрасли и задач сценарного моделирования сложных космических экспедиций [1].

Выше упоминалось, что использование языка программирования Python само по себе закладывает важную возможность развития архитектуры. Речь о том, что в последние годы на этом языке интенсивно разрабатываются программные средства численных и научных расчетов [19]. При соответствующей их адаптации к идеологии рассмотренной архитектуры, можно будет существенно упростить всю архитектуру и вообще удалить из нее пакет Octave, ограничившись одной средой Python.

Потенциально, можно даже избавиться от платформы Java, поскольку на языке Python уже разработаны бесплатный веб-сервер и свои методы удаленного вызова процедур (RPC). Однако на данный момент это не представляется слишком необходимым, так как платформа Java и веб – сервер Apache Tomcat отличаются заметно более высокой популярностью и, как следствие – простотой эксплуатации, отлаженностью программного кода и надежностью.

Результаты, описанные в статье, были ранее частично опубликованы в ИПМ им. М.В. Келдыша РАН [1], а также доложены на двух международных конференциях [20, 21].

Литература

1. Р.Д. Зухба, П.В. Куракин, Г.Г. Малинецкий, С.А. Махов, Н.А. Митин, А.П. Сорокин. “Программно – математические комплексы систем поддержки принятия решений нового поколения”. – Препринт Института прикладной математики им. М. В. Келдыша РАН № 9, 2014 г. – 33 с.

2. МАТЛАБ (статья в Wikipedia) https://ru.wikipedia.org/wiki/MATLAB.
3. Simulink (статья в Wikipedia) https://en.wikipedia.org/wiki/Simulink.
4. Архитектура клиент – сервер (статья в Wikipedia) https://ru.wikipedia.org/wiki/Клиент-сервер.
5. Формат описания структурированных данных JSON (статья в Wikipedia): http://ru.wikipedia.org/w/index.php?title=JSON.
6. Сетевая модель ISO (статья в Wikipedia) https://ru.wikipedia.org/wiki/Сетевая_модель_OSI.
7. HTTP (статья в Wikipedia) https://ru.wikipedia.org/wiki/HTTP.
8. Apache Tomcat (статья в Wikipedia) https://ru.wikipedia.org/wiki/Apache_Tomcat.
9. Сериализация (статья в Wikipedia) https://ru.wikipedia.org/wiki/Сериализация.
10. Yahoo! UI Library (статья в Wikipedia) https://ru.wikipedia.org/wiki/Yahoo!_UI_Library.
11. DWR (статья в Wikipedia) https://ru.wikipedia.org/wiki/DWR.
12. Сервлет (статья в Wikipedia) https://ru.wikipedia.org/wiki/Сервлет_(Java).
13. GNU Octave (статья в Wikipedia) https://ru.wikipedia.org/wiki/GNU_Octave.
14. Jython (статья в Wikipedia) https://ru.wikipedia.org/wiki/Jython.
15. Удалённый вызов процедур (статья в Wikipedia) https://ru.wikipedia.org/wiki/Удалённый_вызов_процедур.
16. Официальный веб-сайт проекта Raphael http://raphaeljs.com/.
17. Система поддержки принятия решений (статья в Wikipedia) https://ru.wikipedia.org/wiki/Система_поддержки_принятия_решений.
18. Java (статья в Wikipedia) https://ru.wikipedia.org/wiki/Java.
19. Программирование и научные вычисления на языке Python https://ru.wikiversity.org/wiki/Программирование_и_научные_вычисления_на_языке_Python
20. Куракин П.В., Малинецкий Г.Г., Митин Н.А., Махов *С.А*. «MATLAB – Based Software for Decision Support Systems». Proceedings of International Conference on Computer Technologies in Physical and Engineering Applications (ICCTPEA 2014). СПб.: IEEE Catalog number CFP14BDA-USB, 2014. Russia, Saint-Petersburg, June 30 — July 4, 2014. С. 93.
21. Куракин П. В., Малинецкий Г. Г., Митин Н. А., Махов С.А., Барыкина М.Н., Зухба Р.Д. «Программно-математические комплексы поддержки принятия решений в космической отрасли». Управление развитием крупномасштабных систем (MLSD'2015): Восьмая международная конференция, 29 сент. - 1 окт. 2015 г, ИПУ им. В. А. Трапезникова РАН.

Куракин Павел Вячеславович. Ведущий инженер Институт проблем управления им. В. А. Трапезникова РАН. Окончил Московский физико-технический институт в 1993 году. Автор 23 печатных работ. Область научных интересов: автоматизация физического эксперимента, автоматизация математического моделирования. E-mail: pvkurakin@yandex.ru