ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 258 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Куракин П.В., Малинецкий Г.Г., Митин Н.А.

Новый взгляд на визуальное проектирование в системах прикладных математических расчетов

Рекомендуемая форма библиографической ссылки: Куракин П.В., Малинецкий Г.Г., Митин Н.А. Новый взгляд на визуальное проектирование в системах прикладных математических расчетов // Препринты ИПМ им. М.В.Келдыша. 2018. № 258. 27 с. doi:10.20948/prepr-2018-258
URL: http://library.keldysh.ru/preprint.asp?id=2018-258

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

П.В. Куракин, Г.Г. Малинецкий, Н.А. Митин

Новый взгляд на визуальное проектирование в системах прикладных математических расчетов

Москва — 2018

П.В. Куракин, Г.Г. Малинецкий, Н.А. Митин

Новый взгляд на визуальное проектирование в системах прикладных математических расчетов

Изложены перспективные предложения по развитию ранее разработанного и испытанного авторами браузерного графического редактора. Изначально редактор создавался как компонент специализированной системы прикладных математических расчетов. Предложены аргументы в пользу своевременности автономного развития редактора. Целевая область применения редактора: системы поддержки принятия решений.

Ключевые слова: автоматизация программирования, моделирование логистики и производства, когнитивное моделирование, специализированные математические расчеты, метаязык, графический редактор, веб-браузер, HTML, JavaScript, JSON, AJAX

P.V. Kurakin, G.G. Malinetskiy, N.A. Mitin

A new look at projecting of visual representation for applied mathematical computation

Perspective concept of a previously designed web browser visual editor is described. Initially the editor intended to be only a component of a large system for specialized applied mathematical computation. Now, arguments for autonomous development of this component are suggested. The target application area for the editor is decision support systems.

Key words: automation of programming, modeling of logistics and industrial production, cognitive modeling, metalanguage, specialized mathematical computation, visual editor, web browser, HTML, JavaScript, JSON, AJAX

Работа выполнена при поддержке РФФИ (проекты 16-01-00342) и программы фундаментальных исследований президиума РАН, проект 3.2 «Разработка фундаментальных основ прогнозирования, экспертизы и принятия управленческих решений в научно-инновационном комплексе России на базе информационного и компьютерного моделирования и когнитивных центров».

Содержание

1. Введение и постановка задачи

Многие программные системы математических расчетов снабжены системами визуального проектирования. В первую очередь, конечно, следует упомянуть систему SIMULINK [1, 2], которая является графическим расширением системы MATLAB [3]. Такое расширение позволяет создавать экземпляры задач визуально, в специфическом графическом редакторе. Структурно описание экземпляра задачи состоит в создании совокупности *блоков* и соединяющих их *линий*. Согласно [2], в системе предусмотрены следующие типы блоков:

– «источники»: используются для генерации всевозможных сигналов;
– «стоки»: используются для вывода или отображения сигналов;
– «дискретные (элементы)»: линейные элементы дискретного времени (передаточные функции, модели пространства состояний и т.п.);
– «линейные (элементы)»: линейные элементы систем непрерывного времени (суммирующие соединения, линейные усилители и т.п.);
– «нелинейные (элементы)»: нелинейные операторы (произвольные функции, линии насыщения и задержки);
– «соединения»: мультиплексоры [4], демультиплексоры [5] и т.п.

SIMULINK – самое известное и популярное программное обеспечение с указанными функциональными возможностями. Имеет смысл упомянуть родственные продукты.

FlowDesigner [6] – «…свободно распространяемая (лицензия GPL/LGPL) среда разработки, ориентированная на потоки данных. Ее можно использовать для создания сложных приложений путем комбинирования небольших типовых (reusable) строительных блоков». Заявлен следующий круг задач, где эффективно применение продукта:

– обработка сигналов;
– обработка звуковых сигналов (и произвольных цифровых сигналов);
– векторное квантование;
– нейронные сети;
– нечеткая логика;
– обработка звуковых сигналов в режиме реального времени (бета-версия);
– линейная алгебра;
– робототехника.

Scilab [7] и расширяющая его система графического проектирования и моделирования Scicos [8]. Согласно [8], «Scicos – независимое программное приложение для моделирования, но доступ к Scilab и его функциональным возможностям обеспечивает большую гибкость и расширяет диапазон возможностей моделирования». О системе Scicos сообщается как о системе моделирования динамических систем, причем основной акцент в системе сделан на моделировании больших потоков данных.

Система Labview ([9, 10]) – "это среда разработки и платформа для выполнения программ, созданных на графическом языке программирования «G» фирмы National Instruments (США)... Labview используется в системах сбора и обработки данных, а также для управления техническими объектами и технологическими процессами". В репертуаре Labview мы имеем дело скорее с некоторым стандартным типом статистических и технологических задач, нежели системой математических расчетов общего назначения.

Общая черта перечисленных программных средств – эти системы объединяют в себе проектирование и моделирование. Под словом «моделирование» мы здесь понимаем автоматическую генерацию программного кода (на том или ином языке программирования), который будет обрабатывать созданный экземпляр задачи, и исполнение этого кода. То есть профессиональные, коммерческие программные продукты описываемого типа предоставляют «полный цикл» подготовки и решения экземпляра прикладной математической задачи того или иного типа: от визуального проектирования этого экземпляра средствами графического редактора до получения автоматического «решения» (в том или ином смысле) этого экземпляра.

Авторам в своей профессиональной деятельности пришлось столкнуться с необходимостью разработки подобной системы, но в условиях минимального наличия финансовых и человеческих ресурсов. Полученное решение описано в [11–13]. Решение опирается на свободно распространяемое программное обеспечение (например, графический пакет Raphael [14], созданный на языке браузерного программирования JavaScript) и не предполагает автоматической генерации программного кода. Решение позволяет создавать экземпляры задач определенных типов; описание созданного экземпляра далее передается в вычислительный модуль; программный код (в нашем случае на языке программирования MATLAB \ Octave) для вычислений разработан специалистом заранее.

Можно указать два типа задач, с которыми пришлось иметь дело авторам:

1. моделирование производства в космической отрасли в смысле потоков стоимостей и поставок комплектующих между предприятиями;
2. моделирование логистики дальних космических полетов в условиях нескольких параллельных стартов.

Схематически архитектура нашего программного решения и направления потоков информации в нем очень простые:

(веб-браузер) ↔ (веб-сервер) ↔ (вычислительный модуль)

Графический редактор – основная тема данной работы – реализован именно в среде веб-браузера, мы продолжим опираться на это узловое решение.

По сути, это стандартная архитектура типа «клиент–сервер», где в качестве “back-end” выступает не СУБД (система управления базами данных), как в большинстве коммерческих приложений, а вычислительный модуль. Сначала в качестве вычислительного модуля мы использовали пакет MATLAB [3], потом перешли на его бесплатный аналог – пакет Octave [14, 15]. В качестве клиента в нашем случае выступает веб-браузер, в среде которого работает графический

редактор упомянутого выше типа: он позволяет создавать экземпляры задач. Сразу отметим (это важно для дальнейшего изложения), что описание (далее – конфигурация) экземпляра задачи поступает из редактора (интерфейсного слоя) в вычисляющий слой в текстовом виде в формате JSON [17].

Мы не ставили задачу автоматической генерации программного кода, но постепенно мы пришли к выводу, что можем ставить другую задачу. Приведенный выше краткий обзор имеющихся на рынке программных решений позволяет отметить, что все они так или иначе (начиная с «родоначальника» этого семейства программ – пакета SIMULINK) рассчитаны на достаточно стандартные, типовые расчетные задачи из различных областей науки и техники. Опыт авторов показал, что довольно часто заказчика из государственных органов интересуют задачи, структура которых может быть приближена стандартными образцами пакета SIMULINK довольно плохо. Речь о том, что сами составные элементы, из которых составляются модели в SIMULINK, как оказалось, плохо приспособлены для целого ряда нишевых задач.

Авторов заинтересовала возможность разработки относительно «малой кровью» такого графического редактора, который можно было бы *гибко настраивать* под задачи разных типов. Вопрос об автоматическом получении программного кода, обрабатывающего создаваемые экземпляры задач, при этом не рассматривается.

Мы предполагаем следующую ситуацию: наш пользователь уже работает с прикладным математиком; этот математик понимает содержательный смысл проблем заказчика и может сам написать нужный обрабатывающий код для каждого *типа* задач.

Но нашему заказчику необходимо создавать много разных экземпляров задачи нужного ему типа. Важно автоматизировать именно эту стадию работы. Это позволит *эффективно разделить труд* нашего пользователя и сотрудничающего с ним математика. Пусть в каждом конкретном случае, то есть – для каждого типа задач, математик сам разработает численный алгоритм. Тип задачи математику разъяснит наш предполагаемый пользователь, он же сам (или при помощи математика) настроит графический редактор для проектирования экземпляров вычислительных задач необходимого типа. Для нас важно, что в результате наш пользователь сможет многократно обрабатывать различные экземпляры задач нужного ему типа.

Таким образом, наша постановка задачи отличается от той, которую традиционно ставят перед собой разработчики пакетов типа SIMULINK. Коммерческие компании закономерно видят свою цель в автоматизации решения типовых научно-технических задач, мы же хотим научиться работать с нишевыми задачами. Мы хотим добиться гибкости визуального проектирования экземпляров задач в графическом редакторе ценой сознательного отказа от автоматической генерации вычислительного кода.

Отметим, что для наших целей, в отличие от предыдущих разработок, достаточно одного веб-браузера. Дело в технологии AJAX [18], которая позволяет

загружать и выгружать текстовые файлы в браузер и из браузера, в том числе прямо из файловой системы, без посредства веб-сервера. Текстовые файлы будут содержать конфигурацию задачи (см. далее).

2. ФОРМАЛЬНАЯ СТРУКТУРА ЭКЗЕМПЛЯРОВ ЗАДАЧ: ПРИМЕРЫ

Рассмотрим конкретные типы задач и посмотрим, какова формальная структура описания экземпляров задач соответствующих типов.

1.

Моделирование производства в космической отрасли в смысле потоков стоимостей и поставок комплектующих между предприятиями. В этом типе задач базовой структурой данных, описывающих задачу в целом, является математический граф. Узлами графа являются предприятия (точнее, *объекты*, отображающие предприятия – относительно небольшие именованные структуры данных), в качестве ребер графа выступают объекты поставок той или иной продукции между предприятиями. Строго говоря, раз мы тоже считаем ребро объектом, надо говорить о том, что такое ребро распадается на узел отдельного вида и два ребра, связывающие этот узел-поставку с узлами-предприятиями. Пока можно опустить эту особенность.

Кроме того, в задаче этого типа предполагалось, что предприятия имеют различную субъектную принадлежность, то есть могут организационно входить в разные объединяющие структуры: холдинги, концерны и т.п. Таким образом, крупномасштабное описание экземпляра такой задачи состоит из *двух* графов: графа поставок (между предприятиями) и графа субъектной принадлежности (предприятий).

В формате JSON текстовая конфигурация задачи будет выглядеть примерно следующим образом.

```
{
    “all_companies”: [“company_1”, “company_2”, “compa-
    ny_3”, “company_4”, “company_5”],

    “all_corporations”: [“corp_1”, “corp_2”, “corp_3”,
    “corp_4”],

    “all_products” : [“product_1”, “product_2”, “prod-
    uct_3”, “product_4”, “product_5”],
“all_productions” : [
        “company_1” : [“product_1”, “product_5”],
        “company_2”: [“product_1”, “product_3”],
        …
        “company_5” : [“product_5”, “product_2”]
```

```
    ],

"all_supplies" : [
        "supply_1" : {
            "source" : "company_1",
            "target" : "company_5",
            "product" : "product_2"
        },
        …
        "supply_3" : {
            "source" : "company_3",
            "target" : "company_3",
            "product" : "product_1"
        }
    ],

    "subordination" : {
        "company_1" : "corp_2",
        "company_2" : "corp_2",
        …
        "company_5" : "corp_4"
    }
}
```

Разберем эту конфигурацию подробней. С точки зрения традиционной программистской терминологии, эта конфигурация представляет собой *отображение* или *словарь* (в англоязычной программистской терминологии – *map*), в котором каждому ключевому слову (кратко – ключу) поставлена в соответствие некая структура данных. Это может быть просто атомарное значение, массив (список) или отображение.

Данная конфигурация содержит следующий набор ключей: `"all_companies"`, `"all_corporations"`, `"all_products"`, `"all_supplies"`, `"subordinations"`. Можно сказать, что конфигурация состоит из разделов с соответствующими названиями.

В разделе `"all_companies"` просто перечислены списком (одномерным массивом) все рассматриваемые копании (предприятия). В разделе “corporations” перечислены, также списком, все рассматриваемые корпорации. Под корпорацией мы имеем в виду любую экономическую структуру, которая может в том или ином юридическом и организационном смысле объединять несколько предприятий. В виде «корпорации» могут выступать такие субъекты хозяйственного права, как концерн, производственное объединение, финансово-промышленная группа и т.п. В разделе “products” перечислены списком все имеющиеся в рассматриваемой системе виды промышленной продукции, которые производят рассматриваемые предприятия и поставляют друг другу.

Раздел "all_productions" представляет собой словарь, в котором ключами являются названия предприятий, и каждому предприятию сопоставлен список производимой этим предприятием видов продукции. Обращаем внимание, что производимую продукцию следует отличать от поставляемой по коммерческим контрактам. Предприятие может производить какой-то вид продукции, но поставлять его вовне того круга предприятий, которые перечислены в конфигурации, то есть – на внешний рынок. Кроме того, какие-то виды продукции могут производиться для внутреннего потребления на самом предприятии.

Раздел "all_supplies" как раз указывает поставки конкретных видов комплектующих между теми предприятиями, которые «внутри системы», то есть теми, которые учтены в данной конфигурации в разделе "all_companies". Каждый элемент, входящий в раздел "all_supplies", представляет собой словарь (отображение), в котором перечислены параметры каждой коммерческой поставки: "source" (источник) – предприятие-поставщик, "target" (цель, мишень) – предприятие-получатель, и "product" – поставляемый вид продукции. Необходимо уточнить, что как предприятие-поставщик, так и предприятие-получатель обязательно должны быть указаны в разделе "companies" общей конфигурации, а элемент "product" – в разделе "all_products".

Последний раздел "subordinations" показывает субъектную принадлежность предприятий. Разумеется, надо иметь в виду, что приведенная конфигурация имеет очень рамочный характер, в ней указаны только простейшие параметры, имеющие отношение ко всей задаче. Ясно, например, что объект «поставка» ("supply") должен нести много информации, детализирующей коммерческие характеристики данной сделки.

Конфигурации, подобные приведенной, визуально будут выглядеть примерно следующим образом. Рис. 1 дает типовой граф поставок.

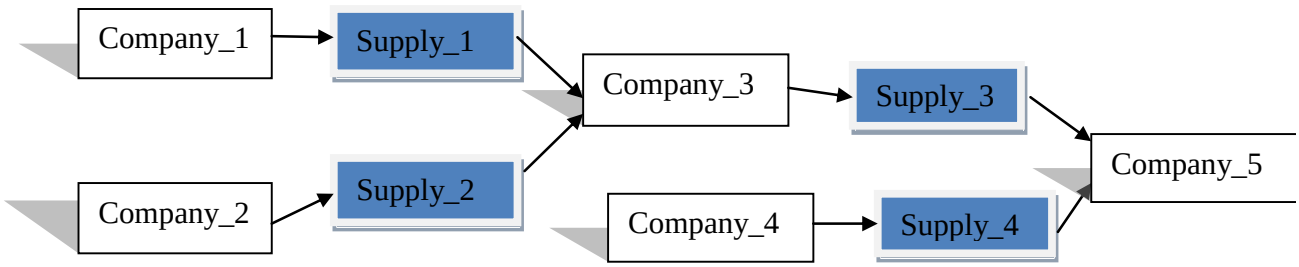

Рис 1. Типовой граф поставок

Рис. 2 дает типовой граф субъектной принадлежности. Со стороны можно бы заметить, что субъектное подчинение предприятий можно было бы выделять цветом прямо на графе поставок. Надо сказать, что поначалу авторы именно так и пытались делать, но опыт эксплуатации полученного графического редактора показал, что разделение информации на два графа более практично и эргономично с точки зрения пользователя.

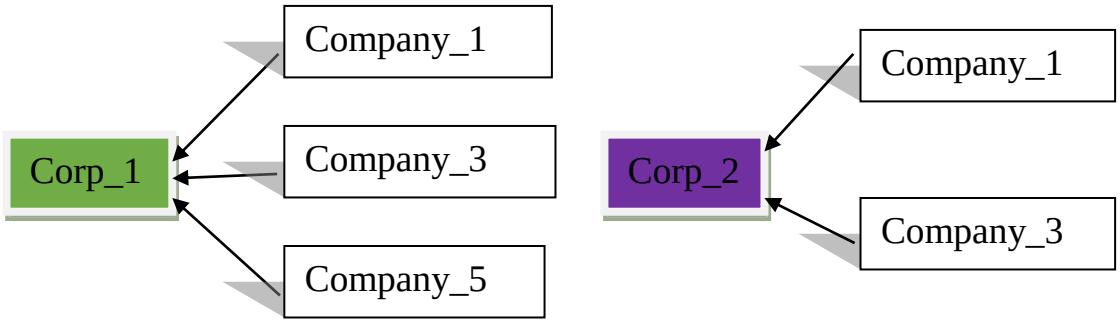

Рис. 2. Типовой граф субъектной принадлежности предприятий

2.

Моделирование полетной логистики сложно организованной космической экспедиции (например, пилотируемый полет на Луну с несколькими наземными стартами – именно для такой задачи была адаптирована версия графического редактора, которую мы описывали в [11]). В этом случае имеются следующие два типа узлов графа: узел полетного события (по согласованию с заказчиком мы называли такие узлы «полетными точками») и узел переходного процесса между событиями (соответственно, мы называли этот узел «полетной операцией»). Так же, как в предыдущем случае, узел переходного процесса появляется, когда на самом деле пользователь имел в виду ребро (стрелку, символизирующую процесс перехода между двумя событиями) графа, но поскольку это ребро само оказывается нагружено данными, то с формальной точки зрения имеет смысл «разорвать» это ребро, сделав из него два «обычных» ребра (не нагруженных данными), между которыми находится узел дополнительного типа. А этот промежуточный узел уже нагружен некоторым набором данных.

Граф, описывающий планируемую логистику всей экспедиции, называется сценарием полета. Сценарий полета может иметь точки ветвления, которые соответствуют возможным вариантам сценария, вызванным, например, возможными поломками либо какими-то наземными обстоятельствами. Фрагмент типового графа сценария полета приведен на рис. 3.

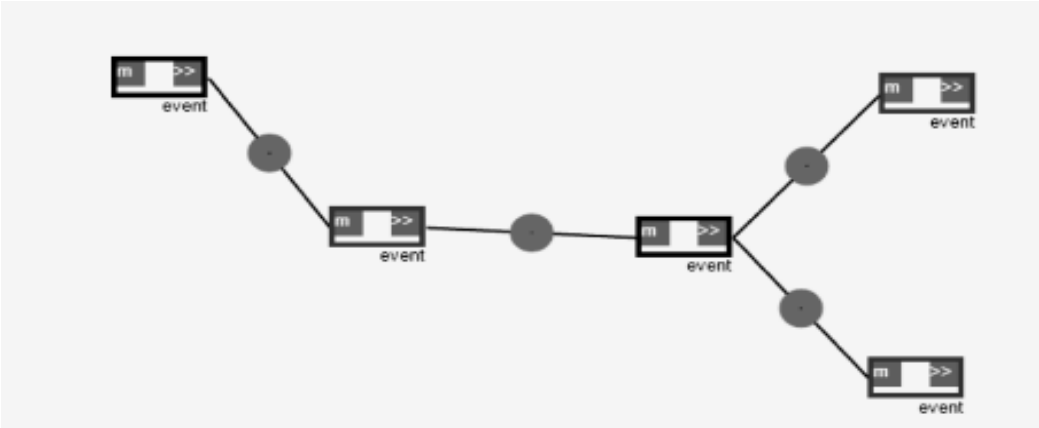

Рис. 3. Фрагмент типового графа сценария космической экспедиции

JSON–конфигурация такого фрагмента будет выглядеть примерно следующим образом:

```
{
    "all_events" : {
        "event_1" : {
            "title" : "orbit_1",
            "height" : 152.34,
            "excentricity" : 1.2
        },
        "event_2" : {
            "title" : "orbit_2",
            "height" : 216.87,
            "excentricity" : 1.5
        },
        …
        "event_5" : {
            "title" : "orbit_5",
            "height" : 176.02,
            "excentricity" : 1.2
        }
    },
    "all_operations" : {
        "operation_1" : {
            "start" : "event_1",
            "end" : "event_2"
        },
        …
        "operation_4" : {
            "start" : "event_3",
            "end" : "event_5"
        }
    }
}
```

Эта конфигурация состоит из двух разделов: "events" (события, полетные точки) и "operations" (полетные операции). Каждый из этих разделов является отображением (словарем). В разделе "events" ключами являются названия полетных точек ("event_1", "event_2" и т.п.). Обращаем внимание, что набор полей (ключей) у каждой полетной точки (точнее, у отображения\словаря, связанного с этой точкой) один и тот же. Но это вовсе не обязательное условие, накладываемое на конфигурацию. Например, для полетной точки "event_3" можно добавить поле "choice_condition" (условие выбора), значение которого будет определять, по какому из двух возможных вариантов ("operation_3" или "operation_4") пойдет дальше сценарий.

Необходимо ли такое поле для каждого события (тогда оно будет иметь непустое значение только в выделенных случаях) или поля можно произвольно добавлять к произвольным словарям – зависит от выбранного языка программирования.

Раздел "`all_operations`" содержит объекты – словари, описывающие все полетные операции. Каждый такой словарь содержит всего два поля, которые, как нетрудно предположить, указывают на начальную и конечную полетные точки, связанные этой полетной операцией. Конечно, мы привели очень упрощенную ситуацию. Очевидно, что указанные поля "`start`" и "`end`" – самая минимальная информация, которую необходимо привести, чтобы описать полетную операцию. На самом деле ясно, что надо указать множество технических характеристик этой операции, по крайней мере – расход топлива и угол тяги двигателя, необходимые для совершения маневра.

3.

Предыдущие две задачи (выражаясь более точно – типы задач, поскольку нам важно различать тип задачи и экземпляр задачи) довольно близки друг другу по духу. Рассмотрим новый, качественно отличный тип задач. Конфигурация и графическое представление этого типа задач оказываются более интересными и нетривиальными.

В работе [19] описана система моделирования макроэкономической динамики целой страны (конкретно – России). Формально – математически – модель представляет собой систему большого (порядка 50) числа нелинейных алгебраических уравнений вида

$$x(i,t) = f(x(1,t), x(2,t), \ldots x(i-1,t), \ldots x(i+1,t), \ldots, x(N,t), a(1,t), \ldots a(M,t)). \quad (1)$$

Здесь $x(i,t)$ – значение *i*-й динамической переменной модели в дискретный момент времени t (время измеряется годами), $a(j,t)$ – значение *j*-го параметра модели в этот год. Различие между переменными и параметрами не несет содержательного экономического смысла. Это способ, которым модель интерпретирует макроэкономическую реальность. И переменные, и параметры суть различные макроэкономические показатели, но временная (по годам) зависимость части из них предполагается заданной извне (как прогнозы экспертов), а зависимость остальных показателей подлежит расчету. Иными словами, по заданным $a(j,t)$ нам надо вычислить $x(i,t)$.

Как может выглядеть графическое описание задач такого типа? В данном случае уже нельзя говорить, что структура модели представляет собой один математический граф, во всяком случае – обычный граф. Уравнение (1) записано в общем виде, а конкретные зависимости из [19], как правило, довольно простые, включают не более 3-4 переменных с простыми алгебраическими операциями (сложение, умножение, деление). Каждая переменная зависит одним

уравнением от небольшого числа других переменных и параметров, но эта зависимость в общем случае не является симметричной, т.е. мы, во всяком случае, не получим один ненаправленный граф.

Нетрудно прийти к выводу, что вообще не имеет смысла пытаться описать задачу одним графом. Вообще, похоже, не стоит пытаться показать пользователю все описание задачи (уравнения) целиком. В текущей реализации задачи вся модель реализована в среде Excel. Это удобно с той точки зрения, что уравнения, исходные данные и получаемые результаты расчетов в виде графиков содержатся в одном файле. Но это не делает описание задачи более обозримым, а работу с ним – более техничной.

Мы решили пойти по следующему пути. Для каждой переменной графический редактор должен отображать две группы связей. Каждая переменная, во-первых, зависит от каких-то других переменных. Типичное уравнение из модели [19]: CN2(t) = n2(t)*(XO(t) + IM(t)) (2): чистые налоги на производство и импорт CN2 пропорциональны сумме выпуска XO в основных ценах и импорта IM, n2 – ставка налогообложения. Таким образом, Переменная CN2 зависит от трех других переменных (n2, XO, IM). Это можно изобразить в виде небольшого графа, как на рис. 4. Этот граф уместно назвать «локальным», поскольку он отображает только малую часть от полного описания экземпляра задачи.

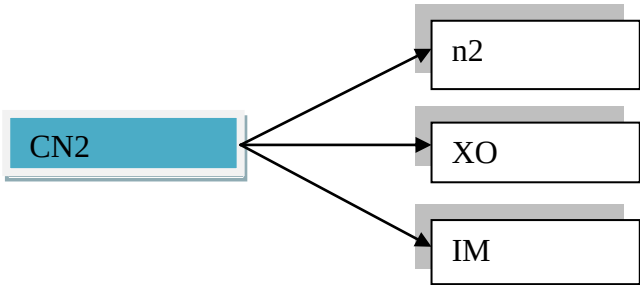

Рис. 4. Локальный граф зависимостей переменной CN2

С другой стороны, от данной переменной CN2 также зависят какие-то другие переменные: например, для номинального ВВП страны WWP в [19] имеется уравнение WWP(t) = WPR(t) + OT(t) + CN2(t) (3), где WPR – произведенный ВВП, OT – оплата труда. Поэтому вторая группа связей, которую имеет смысл показывать в графическом редакторе для выбранной переменной – это все те переменные, которые в своих уравнениях зависят от данной.

Таким образом, одной переменной будут соответствовать *два* локальных графа – с переменными (и параметрами), от которых зависит данная переменная, и с теми, которые, наоборот, зависят от нее. При этом для пользователя было бы весьма удобно, чтобы была организована эффективная навигация между такими локальными графами. Например, наблюдая картину как на рис. 4, пользователь мог бы кликнуть по квадратику “n2” и перейти (на свой выбор) к одному из локальных графов, связанных с переменной n2. Нам представляется, что в таком случае работа с моделью (равно как ее просмотр и ре-

дактирование) становится более удобным и осмысленным, чем в случае одной большой «простыни» с уравнениями в среде Excel.

Возникает закономерный вопрос: в каком виде, где и как хранить уравнения модели, аналогичные (1). В принципе, текстовое представление всех уравнений можно хранить в одном общем файле полной конфигурации задачи. Но в таком случае мы возвращаемся к уже описанной ситуации с моделью [19] – одна большая «простыня» мало пригодна для эффективной практической работы с моделью, особенно если модель большая.

Как нам представляется, существует более эффективный подход. Будем считать, что наш прикладной математик ведет расчеты (и, соответственно, пишет программы) в MATLAB / Octave. Можно быть уверенным, что это хорошее допущение. Тогда каждое уравнение, описывающее динамику одной переменной, следует хранить в отдельном m-файле. У нас может быть целый набор определенным образом структурированных каталогов файловой системы, в которых хранится вся совокупность m-файлов наших уравнений и других (вспомогательных) вычислительных алгоритмов. Для каждого уравнения задачи в файле полной конфигурации достаточно указать только путь к необходимому m-файлу – относительно файла конфигурации, разумеется.

M-файл, соответствующий уравнению типа (2), может иметь название cn2.m (вне зависимости от выбранного программного обеспечения для расчетов лучше никогда не рассчитывать на его чувствительность к регистру и использовать только строчные буквы) и иметь примерно следующий вид:

```
function return = cn2(n2, xo, im)

% чистые налоги на производство и импорт
% n2 – ставка налогообложения
% xo – выпуск в основных ценах
% im – импорт

return = n2 * (xo + im);
```

Сделаем некоторые необходимые пояснения относительно формата и способа использования m-файлов. Один m-файл всегда соответствует ровно одной функции. Функция может возвращать значение любого типа, включая сложные массивы и т.п. Символ возвращаемого значения указан сразу после слова `function` в первой строке m-файла. В нашем случае это локальная переменная `return` – ее область видимости ограничена файлом cn2.m – и это просто действительное число.

Далее, символы аргументов функции вовсе не обязаны совпадать с теми реальными именами переменных, от которых зависит интересующая нас переменная (в данном случае CN2). Конечно, всегда удобнее работать, если совпадение имеет место, но не следует на это полагаться слишком сильно, даже если

сознательно стремиться готовить конфигурацию и сопутствующие m-файлы именно так. Что на самом деле важно, это чтобы число аргументов функции, задаваемой m-файлом, совпадало с количеством переменных (и параметров), с которыми связана исходная переменная – как на рис. 4. Предлагаемому программному обеспечению следует проверять совпадение этих величин, при этом совпадение символов можно игнорировать.

Напомним: нас интересуют не только переменные и параметры, от которых зависит данная переменная, но и *все* те переменные, которые зависят от данной. Можно первый вид зависимостей назвать *прямой* зависимостью, а второй – *обратной*. Следует ли, и если да – то каким образом, хранить в полной конфигурации данные об обратных зависимостях?

На наш взгляд, этого не стоит делать для *статического* представления конфигурации – то есть того вида, как она хранится в JSON–файле. Эту информацию следует сделать *вычисляемой*: на основе имеющейся в статической конфигурации (загруженной из JSON–файла) информации о зависимостях всех переменных алгоритмически получить список всех переменных, зависимых от данной. После получения такого списка его не надо записывать в файл, но можно *кешировать*, т.е. сохранить в памяти для текущего сеанса работы пользователя. В этом случае можно говорить о *динамическом* представлении конфигурации.

Приведем упрощенный пример (статической) конфигурации, соответствующей данному типу задач.

```
{
    "all_vars" : ["x1", x2, x3, x4, x5],
    "all_params": ["p1", "p2", "p3"],
    "all_direct_dependencies" : {
        "x1" : {
            "vars" : ["x2", "x3", "x4"],
            "params" : ["p2"],
            "m-file" : "x1.m"
        },
        "x2" : {
            "vars" : ["x1", "x5"],
            "params" : ["p1", "p3"],
            "m-file" : "x2.m"
        },
        "x3" : {
            "vars" : ["x2"],
            "params" : [],
            "m-file" : "x3.m"
        },
        …
```

```
    }
}
```

В данной конфигурации все достаточно прозрачно и очевидно. Комментария заслуживает разве что тот факт, что переменная x3 вообще не зависит от параметров (они перечислены в разделе "all_params"), однако с формальной точки зрения все равно необходимо указать список параметров, от которых x3 зависит функционально, даже если этот список пустой.

Допустим, пользователь выбрал для просмотра переменную x2. При этом автоматически происходит описанный выше процесс алгоритмического построения списка обратных зависимостей для этой переменной, тогда динамическое представление конфигурации (хранящееся в памяти редактора, т.е. фактически веб-браузера) принимает следующий вид (мы приведем только ту часть, в которой динамическая версия дополняет статическую):

```
{
    …

    "reverse_dependencies" : {
        "x1" : {
            "vars" : ["x2", "x5"],
            "m-file" : "x1.m"
        },
        "x3" : {
            "vars" : ["x1", "x2"],
            "m-file" : "x3.m"
        },
        "x4" : {
            "vars" : ["x3", "x5],
            "function" : "x4.m"
        },
    }
}
```

Обратим внимание на следующее: в статической конфигурации раздел прямых зависимостей называется "all_direct_dependencies", поскольку речь идет в самом деле обо *всех* прямых зависимостях, они все должны быть перечислены в полной (статической, хранящейся в файле) конфигурации. Если мы не добавляем новые уравнения в модель, то структура статической конфигурации не изменяется. Хотя мы можем изменить то или иное уравнение, но сама зависимость данной переменной от других остается, мы в принципе не можем обойтись без ее описания. В динамической конфигурации раздел обратных зависимостей называется "reverse_dependencies", здесь речь не

идет обо *всех* зависимостях. Эти зависимости, как указано выше, система вычисляет и помещает в динамическую конфигурацию постепенно, «по требованию», то есть по мере работы пользователя с графическим редактором. В общем случае в динамической конфигурации в разделе "`reverse_dependencies`" во время сеанса работы пользователя хранятся только некоторые из обратных зависимостей.

С другой стороны, можно рассмотреть такой вариант, когда пользователь может сохранить в файл динамическую версию конфигурации, чтобы потом воспользоваться результатами предыдущего сеанса собственной работы. И таких файлов может быть несколько. В этом случае надо обсуждать систему наименования или какого-то индексирования уже целой совокупности связанных конфигурационных файлов.

Отметим, что если иметь в виду только что рассмотренный тип задач, то реализация задуманного нами графического редактора отвечала бы целям, заявленным в [20].

4.

Задачи когнитивного моделирования [21]. Эти задачи актуальны, например, при комплексном анализе региональных проблем. Часть авторов данной работы участвовала в формулировке предложений по так называемым когнитивным центрам [22]. В этой области базовой моделью традиционно является также граф, в данном случае ориентированный. Вершины этого графа соответствуют некоторым факторам влияния, выделенным исследователем исходя из своего понимания рассматриваемой сложной системы, а ребра – взаимосвязям между этими факторами. В типичном случае цикла моделирования по системе запускают импульс, соответствующий изменению одного из факторов, и смотрят: как это повлияет на все остальные факторы. Примерный вид типичного когнитивного графа приведен на рис. 5, позаимствованном из статьи [23].

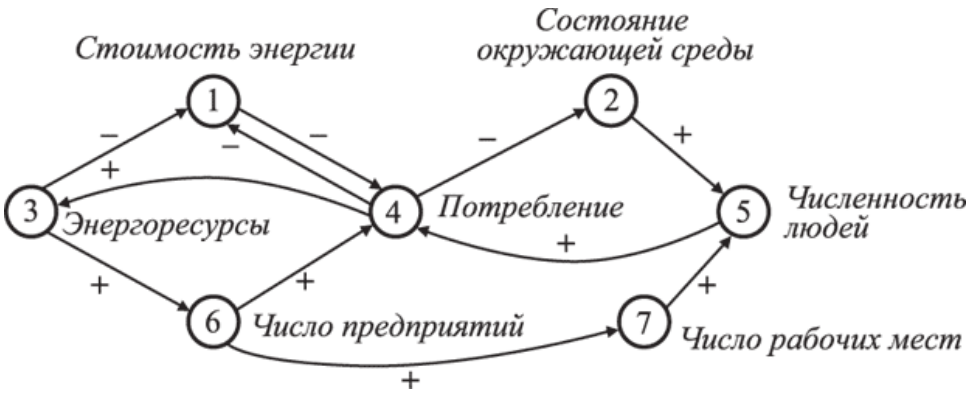

Рис. 5. Типовой когнитивный граф сложной системы

Для нас наибольший интерес в этой сфере представляют именно слабоструктурированные и слабо формализованные задачи. А это значит, что сама структура соответствующего «когнитивного графа» может непрерывно меняться пользователем непосредственно в процессе работы – исследователь экспе-

риментирует с разными факторами и связями между ними и их параметрами. В этой ситуации была бы крайне желательна возможность оперативной «навигации» между близкими когнитивными графами с возможностью сравнения результатов моделирования в смежных структурных моделях. При этом следует ожидать, что «соседние» варианты когнитивных графов отличаются наличием или отсутствием совсем небольшого количества дополнительных узлов и\или ребер. Строго говоря, имеет смысл добавлять, удалять, или изменять содержательное наполнение (смысл количественных параметров, приписанных узлу или ребру) всего одного узла или ребра. Иными словами, в системе вариантов задачи следует ожидать, что смежными будут варианты, отличающиеся на «элементарные» шаги. Следует также ожидать, что такая система связанных вариантов сама по себе представляет собой некий граф – возможно, не являющийся плоским. Желательно иметь возможность также отображать этот граф визуально.

В целом возможность построения такого графа связанных вариантов когнитивного графа могла бы заметно расширить возможности исследователей сложных систем. Авторам неизвестно наличие в настоящее время инструментов подобного рода.

Для системы, когнитивный граф которой приведен на рис. 5, можно предположить следующее «элементарное» изменение: добавим фактор (соответственно, узел) «Социально-экономическая политика администрации региона», который будет связан с двумя уже существующими факторами «Численность людей» и «Потребление».

Предполагается, что «хорошая» (активная, эффективная) социально-экономическая политика увеличивает численность населения (допустим, независимо от числа рабочих мест, что может выглядеть странно, но для условного примера вполне приемлемо). При этом сама эта политика пусть зависит, допустим, от фактора «Потребление». Имеется в виду, что чем выше уровень потребления, тем выше культурные запросы и образовательный уровень населения, что влияет на более высокую разборчивость людей при выборах региональной администрации.

Обсудим, как может выглядеть JSON–конфигурация задач такого типа. Сразу отметим, что JSON–описание графа, тем более целой совокупности графов, можно оформить и структурировать не единственным образом. Каждое из них имеет собственные достоинства и недостатки, каждое по-своему как «эффективно», так и «не эффективно».

Например, с учетом выше сказанного, экземпляр задачи можно описать непосредственным перечнем все тех же узлов и ребер графа, но каждому такому элементу необходимо дополнительно приписать перечень вариантов задачи, в котором данный элемент (узел или ребро) присутствует. Кроме этого, имеет смысл внести в полную конфигурацию формальное описание «графа связности графов».

```
{
    ”all_factors” : {
        “Energy_cost” : [“System_1”, “System_2”],
        “Energy_resources” : [“System_1”, “System_2”],
        …
        “Gvmnt_policy” : [“System_2”]
    },
    “all_links” : [
        [“Energy_resources”, “Energy_cost”, “System_1”,
        “System_2”, “-”],
        …
        [“Consumption”, “Gvmnt_policy”, “System_2”,
        “+”],
        [“Gvmnt_policy”, “Population”, “System_2”, “+”]
    ]
}
```

В этой конфигурации связи влияния, относящиеся только ко второму варианту (те, что присутствуют только в “System_2”), выделены курсивом.

Все сведения о каждом ребре графа (т.е. связи влияния между факторами) перечисляются в одном одномерном массиве (списке): сначала указан первый элемент связи (начало ребра – мы имеем дело с направленным графом), затем второй элемент связи (конец ребра), далее перечислены варианты задачи («системы»), в которых присутствует данная связь, в конце указан знак данной связи. Сами списки также объединены в один общий список.

Такой способ описания связи удобен своей компактностью, но он имеет свои недостатки. Сразу можно сказать, что линейный список из разнородных элементов (разных типов данных) изначально выглядит подозрительно. Далее предполагается, что в каждом варианте (если указаны несколько) связь имеет один и тот же знак. Между тем варианты могут отличаться как раз тем, что одна и та же связь принимает в них разные знаки.

В принципе, в том случае, если это происходит, можно разбить один список на два, для каждого варианта задачи:

```
{
    …
    “all_links” : [
        …
        [“Environment”, “Population”, “System_1”, “+”],
        [“Environment”, “Population”, “System_2”, “-”],
        …
    ]
}
```

Однако такое решение тоже содержит свои подводные камни: обрабатывающая программа заранее не может знать, какой именно способ перечисления свойств связи используется, поэтому понадобится, по крайней мере, проверка длины списка и условная точка ветвления программы для обработки содержания списка.

Либо имеет смысл делать в виде списка описание каждого «ребра–связи» для каждого варианта. Надо отдавать себе отчет в том, что это увеличит размер файла конфигурации. С точки зрения работоспособности программы, это не имеет большого значения. Однако, напомним, мы решаем не просто техническую или технологическую задачу. Для нас большое значение имеет *эргономичность* системы, удобство пользователя при работе с ней. Мы предполагаем, что наш пользователь может работать не только с визуальным представлением описания своей задачи, но и с текстовым. Часто может оказаться гораздо проще внести правку в текстовый файл, чем загружать его в графический редактор. Поэтому размер текстового представления полной конфигурации задачи тоже имеет значение. Точнее, важно, чтобы этот текст был *удобочитаемым* – в принципе, это может быть достигнуто при удобной структуре текста, и при большом размере этого текста. Поэтому вопросы выбора формата конфигурации имеют свое прикладное значение.

Можно поступить следующим образом: в явном виде перечислять описания «ребер–связей» для каждого варианта задачи (в принципе, это же верно и для «узлов–факторов» когнитивного графа, просто в этом случае все гораздо проще, здесь не возникает описываемого комплекса проблем и неоднозначностей; на наш взгляд, приведенный выше вариант описания факторов вполне приемлем).

```
{
    …
    "all_links" : {
        "System_1" : [
            ["Energy_resources", "Energy_cost", "-"],
            …
            ["Environment", "Population", "+"]
        ],
        "System_2" : [
            ["Energy_resources", "Energy_cost", "-"],
            …
            ["Environment", "Population", "+"],
            ["Consumption", "Gvmnt_policy", "+"],
            ["Gvmnt_policy", "Population", "+"]
        ],
    }
}
```

Видно, что в этом случае у нас нет необходимости в списке описания каждого «ребра–связи» указывать перечень вариантов («систем»), что при предыдущем способе описания было неизбежно, и мы многократно дублировали строки. Однако теперь, сэкономив на этих строках, мы получили дублирование целых списков (хотя и сокращенных) для тех из них, которые входят во все или хотя бы в несколько вариантов задачи. Решать вопрос о предпочтительности того или иного способа описания надо в каждом конкретном случае. Поскольку мы ставим задачу разработки некоторого языка форматирования функциональных возможностей нашего графического редактора, то этот язык должен включать опцию задания указанных вариантов описания семейства когнитивных графов. В идеале речь должна идти об автоматизированном и оперативном переключении между разными режимами записи конфигурации.

До сих речь шла о статической конфигурации, хранимой в файле. Выше было отмечено, что во время работы программы в оперативной памяти компьютера существует и динамическое представление конфигурации. Предполагалось, что это представление имеет большее значение для самой программы, нежели для пользователя. Выскажем некоторые соображения о связи между двумя представлениями.

Нетрудно заметить, что, с точки зрения динамического представления конфигурации в оперативной памяти, имеет значительный смысл хранить скорее не перечень полных описаний всех вариантов задачи (что и есть полная конфигурация), сколько всего один, текущий вариант и перечень *отличий*, которыми связан этот текущий вариант со смежными модификациями этого варианта. Для JSON–описаний таких отличий можно предложить термин *дифференциальная конфигурация* или *дифференциальное описание*. Выглядеть оно будет примерно так:

```
{
    “System_1” : {

        “description_type” : “full”,
        “variation_from” : “none”,
        ”all_factors” : [
            “Energy_cost”,
            …
            “Population”
        ],
        “all_links” : [
            [“Energy_resources”, “Energy_cost”, “-”],
            …
            [“Environment”, “Population”, “+”]
        ]
    },
```

```
    "System_2" : {
        "description_type" : "variant",
        "variation_from" : "System_1",
        "remove_factors" : [],
        "add_factors" : ["Gvmnt_policy"],
        "remove_links" : [],
        "add_links" : [
            ["Consumption", "Gvmnt_policy", "+"],
            ["Gvmnt_policy", "Population", "+"]
        ]
    }
}
```

Полагаем, что приведенная конфигурация не нуждается в подробных пояснениях. Отметим лишь, что для каждого варианта мы указываем, является ли описание полным или дифференциальным, в последнем случае указываем, от какого другого варианта заданы изменения; для дифференциального описания указываем список конкретных изменений, т.е. удаленных или добавленных «узлов–факторов» и «ребер–связей».

Дифференциальные конфигурации имеют смысл постольку, поскольку мы предполагаем, что наш пользователь работает итеративно: сначала он создает некий стартовый вариант когнитивного графа, а последующие модификации получаются, как правило (это уже было отмечено выше), добавлением, удалением или изменением всего одного элемента («узла–фактора» или «ребра–связи»). Предположительно, точно так же пользователь работает не только при создании новых вариантов, но и последующих сеансах работы и взаимодействии с различными вариантами: он производит последовательную навигацию между смежными, т.е. мало отличающимися, вариантами.

Тогда оказывается, что сохранение дифференциальных конфигураций (JSON–описаний отдельных вариантов) имеет смысл не только в динамическом, но и в статическом представлении полной конфигурации (т.е. не только в памяти программы, но и в файле, сохраняемом в файловой системе).

Можно предположить такую схему работы программы. Полная статическая конфигурация хранит одновременно как перечень полных описаний всех вариантов, так и перечень отличий между локально связанными вариантами (дифференциальные конфигурации вариантов). Причем, что значит «локально связанные» варианты, невозможно определять автоматически – эта связанность создается последовательной работой самого пользователя. Здесь, по аналогии с теорией баз данных, уместно ввести понятие *целостности данных*: каждое описанное в конфигурации *отличие* одного варианта когнитивного графа от «предыдущего» должно приводить в точности к тому же *полному* описанию этого варианта, присутствующему в этом же JSON–тексте полной конфигура-

ции. Также следует ожидать, что самого пользователя при обращении к этому тексту могут интересовать как полные описания, так и дифференциальные.

Для поддержки указанной целостности данных имеет смысл ввести программную процедуру так называемой *валидации* полной конфигурации, которая будет проверять эквивалентность полных и дифференциальных описаний. Эту процедуру можно запускать как автоматически при начале сеанса работы пользователя и загрузки им в браузер полной конфигурации, так и независимо. Надо иметь в виду, что, если пользователь захочет изменить ранее созданный вариант когнитивного графа, но сохранить его не как новый вариант, а как «старый», под тем же названием или идентификатором, необходимо заново пересчитывать все связанные с ним дифференциальные описания.

На первый взгляд представляется, что этот пересчет является не только необходимым, но и достаточным: если мы изменили и сохранили *под прежним названием* какой-то вариант, то необходимо обновить лишь информацию об отличиях со смежными вариантами. Однако приходится вспомнить, что пользователь может осуществлять навигацию между вариантами своего когнитивного графа *в обе стороны*. Следовательно, для каждой пары смежных вариантов придется сохранять (или оперативно вычислять в программе) сразу *две* дифференциальных конфигурации. Судя по всему, достаточно, чтобы эти две конфигурации были связаны простым соотношением: значения полей `"remove_factors"` и `"add_factors"` и, соответственно, `"remove_links"` и `"add_links"` в этих конфигурациях просто взаимно поменяны местами. Возможно, на практике ситуация окажется более деликатной, но мы не будем здесь обсуждать эти детали, поскольку принцип ясен, а практическая реализация еще впереди.

3. Форматирование типа редактора

Мы рассмотрели несколько типов вычислительных задач и обсудили лишь часть структуры их конфигураций – ту, которая предназначена для передачи из графического редактора в вычислительную подсистему. Вопросы визуального представления были рассмотрены в минимальной степени.

Напомним, что мы ставили задачу разработки, в конечном итоге, в широком смысле *универсального* графического редактора, что предполагает создание некоторого «языка» форматирования *типа* редактора. Причем форматированию подлежит как структура данных, которые редактор создает на «выходе», так и визуальное представление создаваемого пользователем экземпляра задачи нужного ему типа.

Обсудим подходы к решению этой задачи.

Во всех типах задач мы имеем дело, так или иначе, с набором некоторых численных переменных, которые надо тем или иным способом сгруппировать и передать прикладному математику, чтобы он подставил их в те или иные урав-

нения. Конфигурации экземпляров задач любого типа состоят, в общем случае, из вложенных друг в друга отображений (словарей), т.е. структур данных типа перечней пар «ключ–значение». Значения могут быть атомарными (скалярными или строковыми), списочными или сами представлять из себя словари. Списки также могут состоять из скаляров, словарей и списков. В конечном итоге, все словари и списки состоят именно из тех переменных, которые нас интересуют в данном типе задач. Вопрос «лишь» в том, как эти переменные распределены по системе вложенных словарей и списков.

Задачу форматирования графического редактора можно условно разбить на три подзадачи. Создавая тип редактора, мы должны, *во-первых*, описать наш набор переменных, причем он не обязательно может быть фиксированным и даже заранее известным. Все, что можно утверждать априори, это что пользователь так или иначе представляет себе общую структуру всей совокупности переменных. Далее, *во-вторых*, нам необходимо неким образом описать, как наши переменные по мере формирования пользователем экземпляра задачи «выкладываются» в определенную структуру словарей и списков. При этом мы имеем в виду полные конфигурации, дифференциальные конфигурации надо обсуждать отдельно. Наконец, *в-третьих*, необходимо описать графическое представление задачи: с какими переменными и как будут связаны узлы и ребра математического графа – ведь мы уже выяснили, что всегда визуально нам нужен именно граф с определенным наполнением его элементов параметрами.

Для начала мы бы предложили следующий метаязык форматирования типа редактора – то есть редактора, умеющего создавать экземпляры задач определенного типа и выгружать их конфигурацию в виде текстового JSON–файла в локальную файловую систему. Этот язык (точнее, его прототип), как нам представляется, объединяет все три подзадачи. Возможно, мы упрощаем ситуацию, и потребуются дальнейшие исследования, но даже чтобы впоследствии отбросить данную идею как слишком наивную, ее надо сначала опробовать.

Базовая идея состоит в следующем. Если присмотреться к приведенным в предыдущем разделе конфигурациям, то можно заметить, что в любой из них тот или иной вложенный блок данных (будь то список, т.е. одномерный массив, или словарь, или атомарное значение) соответствует узлу в графическом представлении этой конфигурации. Разберем на конкретных примерах, повторяя нумерацию предыдущего раздела. Разумеется, все это только в первом приближении.

1. ***Моделирование производства*** в смысле потоков стоимостей и поставок комплектующих между предприятиями. Узлам предприятий соответствуют элементы списка `"all_companies"`. То есть любой квадратик предприятия на Рис. 1 соответствует элементу именно этого массива. С другой стороны, квадратику поставки соответствует словарь (со всем своим содержимым) раздела `"all_supplies"`.

2. ***Транспортная логистика сложной космической экспедиции***. Полетному событию соответствует словарь в составе раздела "all_events", а полетной операции – словарь в составе раздела "all_operations".

3. ***Моделирования макроэкономической динамики***. Квадратиками (узлами) «локальных» графов прямых зависимостей переменных будут словари из раздела "all_direct_dependencies", а также элементы списка "vars", который входит в указанный раздел. Точно так же для обратных зависимостей квадратиками–узлами будут словари из раздела "reverse_dependencies" и элементы поля "vars".

4. ***Многовариантные когнитивные графы***. Здесь узлам соответствуют (атомарные!) элементы списков "all_factors", а ребрам – списки "all_links".

Модель (тип) редактора описывается перечнем всех типов блоков данных, которые могут возникнуть в полной конфигурации задач желаемого типа, с указанием того, какие типы блоков вложены в данный тип и с каким типом блоков может быть соединен ребром данный блок:

```
{
    <block_type> : {
        "key" : <key_name>,
        "value"  :  <value_type>  ("NUMBER",  "STRING",
        "LIST", "MAP"),
        "outer_block" : <block_type>,
        "can_link_block" : <block_type>
    },
    …
    <block_type> : {
            …
    }
}
```

Каждый тип блока как-то называется – <block_type>. Обращаем внимание: это не название блока в реальной конфигурации. В реальной конфигурации данный блок может появиться под названием "key", причем это название может иметь как обязательное (фиксированное), так и произвольное значение. Например, в задачах первого типа название словаря "all_supplies" – обязательное (мы, пользователь и математик, так решили – иначе мы не сможем согласованно передавать описание экземпляра задачи в вычислительный модуль), а вот названия конкретных поставок – произвольные для каждого экземпляра задачи. По умолчанию будем считать значение "key" обязательным, для произвольного значения будем использовать какую-то фиксированную

строку, например – “ANY_STRING”. Если названия у блока вообще нет – будем использовать “NONE”.

Поле “value” – тип данных данного блока – число, строка, список или словарь. Поле “outer_block” – название (<block_type>) типа блока, в который может быть вложен данный блок. Последнее поле – название (“can_link_block”) типа блока, *с графическим представлением* которого *может* быть соединен ребром данный блок. Если поле блока имеет значение “NONE” – данный блок не имеет графического представления.

Приведем фрагмент примерной модели для задачи третьего типа (макроэкономическая динамика):

```
{
    “all_vars” : {
        “key” : “all_vars”,
        “value” : “LIST”,
        “outer_block” : “ROOT”,
        “can_link_block” : “NONE”
    },
    “var” : {
        “key” : “NONE”,
        “value” : “NUMBER”,
        “outer_block” : “all_vars”,
        “can_link_block” : “var”
    },
    …
}
```

Из фрагмента видно, что квадратики визуального представления экземпляра задачи соответствуют переменным. Сами переменные входят в список “all_vars” – то есть это блоки типа “var”. Они могут соединяться с другими переменными, то есть – экземплярами блоков “var”.

Разумеется, это только первоначальная «прикидка» языка форматирования, которая требует дальнейшей проработки.

4. Заключение

Мы считаем, что проведенное исследование позволяет по-новому взглянуть на возможности визуального проектирования в системах математического моделирования и прикладных математических расчетов. Существует достаточно много нишевых прикладных задач, для которых готовые профессиональные и коммерческие системы математического моделирования с подсистемами визуального проектирования оказываются недостаточно пригодными, низко эффективными. Это происходит в силу того, что графические примитивы в визу-

альных редакторах коммерческих продуктов ориентированы хоть на широкие, но, тем не менее, некие стандартизированные наборы научно-технических задач.

Мы показали, как мы полагаем, что существует путь малозатратной разработки систем визуального проектирования с ограниченными возможностями, позволяющих, тем не менее, предоставить широкому классу пользователей-предметников инструментарий для эффективного взаимодействия с сообществом прикладных математиков и математического моделирования актуальных для них задач. Это путь создания графического редактора, настраиваемого пользователем сугубо под его конкретные – часто нестандартные – задачи.

Принципиальная новизна и отличие проектируемого авторами решения от уже существующих в области визуального проектирования для задач прикладной математики – использование веб-технологий. Это соответствует мировым тенденциям информационных технологий в области пользовательских интерфейсов и открывает качественно новые возможности в использовании продукта, в первую очередь – в способах доступа к нему.

Проведенный в работе анализ доведен до предложения принципиальных решений формулировки практических шагов по разработке программного обеспечения желаемого типа.

Библиографический список

1. Simulink (статья в Wikipedia): https://en.wikipedia.org/wiki/Simulink.
2. Simulink basic tutorial. https://classes.soe.ucsc.edu/cmpe242/Fall10/simulink.pdf
3. *Дьяконов В. П.* Справочник по применению системы PC MATLAB. — М.: «Физматлит», 1993. — 112 с. — ISBN 5-02-015101-7.
4. Мультиплексор (статья в Wikipedia), URL: https://ru.wikipedia.org/wiki/Мультиплексор_(электроника)
5. Демультиплексор (статья в Wikipedia): https://ru.wikipedia.org/wiki/Демультиплексор
6. FlowDesigner (страница проекта): https://sourceforge.net/projects/flowdesigner/
7. *Алексеев Е.Р., Чеснокова Е.А., Рудченко Е.А.* Scilab: Решение инженерных и математических задач. Москва ALT Linux; БИНОМ. Лаборатория знаний. 2008 г.
8. *Данилов С.* SCICOS – пакет SCILAB для моделирования динамических систем: http://www.tstu.ru/book/elib2/pdf/2011/danilov.pdf. (Документ выложен на сайте Тамбовского государственного технического университета).
9. Labview (статья Wikipedia), URL: https://ru.wikipedia.org/wiki/LabVIEW.
10. Официальная страница проекта Labview на русском языке, URL: http://www.labview.ru.
11. *Зухба Р.Д., Куракин П.В., Малинецкий Г.Г., Махов С.А., Митин Н.А., Торопыгина С.А.*Система моделирования «КОСКОН» как инструмент поддержки принятия решений в космической отрасли // Препринты ИПМ им.М.В.Келдыша. — 2015. — № 113. — 36 с. — URL: http://library.keldysh.ru/preprint.asp?id=2015-113.
12. *Куракин П.В.* Новая программная архитектура для специализированных систем математических расчетов // Информационные технологии и вычислительные системы, 2016 г, № 2, С. 66 – 74.

13. *Куракин П.В.* Специализированные системы математических расчетов нового поколения // Программные системы и вычислительные методы – 2016 г, №1(14), С. 80 – 94. DOI: 10.7256/2305-6061.2016.1.17997.

14. Официальный веб-сайт проекта Raphael, URL: http://raphaeljs.com/

15. Octave (статья в Wikipedia), URL: https://ru.wikipedia.org/wiki/GNU_Octave.

16. Octave (официальная страница проекта), URL: https://www.gnu.org/software/octave/

17. Официальная домашняя страница формата JSON на русском языке, URL: http://json.org/json-ru.html

18. *Дейв Крейн, Эрик Паскарелло, Даррен Джеймс*. AJAX в действии: технология — Asynchronous JavaScript and XML = Ajax in Action. — М.: Вильямс, 2006. — С. 640. — ISBN 1-932394-61-3.

19. *Антипов В.И., Митин Н.А., Пащенко Ф.Ф*. Макроэкономическая имитационная модель развития России // Препринты ИПМ им. М.В.Келдыша. 2017. № 142. 48 с. doi:10.20948/prepr-2017-142. URL: http://library.keldysh.ru/preprint.asp?id=2017-142

20. *Гусев В.Б., Куракин П.В.* Инструментальная поддержка моделирования динамики саморазвивающихся систем / Седьмая Всероссийская научно-практическая конференция «Имитационное моделирование. Теория и практика» (ИММОД-2015). Труды конференции. 21 – 23 октября. Под общей редакцией академика РАН С. Н. Васильева, чл.-корр. РАН Р. М. Юсупова 2015 г. Москва. Т. 1., С. 63-67.

21. *Максимов В.И., Корноушенко Е.К.* Аналитические основы применения когнитивного подхода при решении слабоструктурированных задач // Труды ИПУ, вып.2, 1998.

22. *Десятов И.В., Малинецкий Г.Г., Маненков С.К., Митин Н.А., Отоцкий П.Л., Ткачев В.Н., Шишов В.В.* Когнитивные центры как информационные системы для стратегического прогнозирования // Препринты ИПМ им. М.В.Келдыша. 2010. № 50. 28 с. URL:http://library.keldysh.ru/preprint.asp?id=2010-50

23. Когнитивный анализ и моделирование в стратегическом управлении. URL: https://studopedia.su/10_104996_kognitivniy-analiz-i-modelirovanie-v-strategicheskom-upravlenii.html