Объектно-ориентированный каркас для программной реализации приложений теории расписаний

[1] *А.С. Аничкин <anton.anichkin@ispras.ru>*
[1,2] *В.А. Семенов <sem@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] *Московский физико-технический институт,*
141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Статья адресована вопросам программной реализации моделей, методов и приложений теории расписаний с использованием объектно-ориентированного каркаса. Каркас представляет собой систему классов вместе с предусмотренными механизмами взаимодействия и расширения, что обеспечивает эволюционную разработку серий приложений на единой методологической, программной и инструментальной основе. В статье детально обсуждаются принципы организации и функционирования разработанного каркаса, а также его возможности для разработки приложений теории расписаний и, в частности, перспективных систем календарно-сетевого планирования и управления проектами.

Ключевые слова: теория расписаний; календарно-сетевое планирование; программная инженерия; объектно-ориентированное программирование.

DOI: 10.15514/ISPRAS-2017-29(3)-14

Для цитирования: Аничкин А.С., Семенов В.А. Объектно-ориентированный каркас для программной реализации приложений теории расписаний. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 247-296. DOI: 10.15514/ISPRAS-2017-29(3)-14

1. Введение

Теория расписаний и календарно-сетевое планирование находят широкое применение в научных и индустриальных областях, связанных с управлением производством, организацией транспортных потоков, управлением вычислительными ресурсами. Однако многообразие существующих математических моделей и вычислительных методов, а также перманентное появление новых ставит перед программистами

довольно острую проблему эволюционной разработки серий приложений на единой методологической, программной и инструментальной основе. В частности, подобная проблема возникает при создании перспективных систем календарно-сетевого планирования и управления индустриальными проектами, в которых задачи составления расписаний решаются в обобщенной постановке с учетом множества факторов, влияющих на ход выполнения проектных работ. В таких постановках учитываются не только типовые временные условия, отношения предшествования между работами, ресурсные ограничения, но и специфические требования пространственно-временной согласованности проектных работ, их финансового и логистического обеспечения. Данные требования существенны для масштабных индустриальных программ, в которых риски технологических и организационных ошибок чрезвычайно высоки, а сроки и бюджеты жестко ограничены. Примерами специфических требований могут служить условия привлечения инвестиционных средств, ограничения по поставкам материалов, правила размещения и использования оборудования, особенности монтажа элементов конструкций возводимого сооружения, условия резервирование рабочих зон при организации проектных работ.

Использование универсальных математических библиотек для решения подобных задач, как правило, оказывается невозможным из-за принципиальных отличий в их постановках или крайне неэффективным в силу зависимости вычислительной сложности составления расписания от частных условий. Например, задача проектного планирования проектов (Resource-Constrained Project Scheduling Problem; RCPSP), являющаяся NP-полной, редуцируется к частным постановкам «открытой линии», «рабочего цеха» или «потоковой линии», имеющим полиномиальную сложность при небольшом числе машин, простых моделях обслуживания и отсутствии директивных сроков [1]. Применение универсальных средств проектного планирования для подобных частных постановок было бы чрезмерно затратным.

Стратегия разработки программ составления расписаний заново для каждого нового типа приложения также является неприемлемой в силу сложности современных математических моделей и вычислительных методов. Данные методы должны учитывать большое количество факторов и использовать развитые системы эвристик для поиска приемлемых приближенных решений в тех случаях, когда задача имеет высокую размерность. Разработка и программная реализация подобных методов часто оказывается предметом интенсивных научных исследований. Адаптация унаследованных открытых кодов, изначально непредназначенных для подобных целей и не предусматривающих возможности для их развития, также малоэффективна даже для написания программ близкой функциональности.

В связи с этим актуальным представляется создание единой инструментальной среды для программной реализации моделей, методов и

приложений теории расписаний. Подобная среда должна предоставлять развитые средства для разработки новых программ на основе ранее реализованных модулей. При этом возможности развития, адаптации и конфигурации модулей должны обеспечивать построение эффективных программ составления расписаний, релевантных условиям и сложности решаемых прикладных задач.

Ранее предпринимались попытки организации подобных сред в виде расширяемых математических библиотек. В качестве одного из значимых результатов следует упомянуть библиотеку PSPLIB [2], в которой Колиш и Шпрехер реализовали несколько методов проектного планирования. Библиотека претерпела определенное развитие [3, 4, 5, 6], однако в настоящее время она преимущественно используется для тестирования других аналогичных программ и оценки производительности на специально подготовленных наборах контрольных примеров (benchmarks). Библиотека имеет принципиальные ограничения для решения индустриально значимых задач высокой размерности, а ее архитектура плохо приспособлена для реализации новых моделей и методов.

Более интересной в этом отношении является библиотека проектного планирования LibRCPS, разработанная Лемменем [7]. В ней предусмотрены интерфейсы для задания входных данных, а также имеется возможность специфицировать некоторые алгоритмические детали поиска решения, например, выбрать тип целевой функции и применяемую эвристику. К сожалению, значительная часть планируемых возможностей и функций библиотеки осталась нереализованной. Несмотря на открытые исходные коды, библиотека не получила дальнейшего развития, а сообщения об опыте ее применения крайне скудны.

Более успешными в практическом отношении оказались программные системы календарно-сетевого планирования и управления проектами. Обычно они обеспечивают автономную работу пользователя на изолированном компьютере или групповую работу в корпоративной сети. В качестве популярных программных решений следует указать Oracle Primavera, MS Project, Synchro, Spider Project, Gemini, Merlin, Zoho Projects, ManagePro [8]. Ряд систем конфигурируется в виде универсальных Интернет-сервисов и WEB-клиентов к ним. К подобным решениям относятся Smartsheet, GanttPro, Asana, Acunote, Teamweek, Bitrix24, Jira, ISETIA [8]. Несмотря на наличие программных интерфейсов доступа к данным и возможность локального или удаленного вызова функций планирования, перечисленные системы обладают существенным общим недостатком. Главным образом он связан с предопределенным характером реализованных алгоритмов и невозможностью их модификации для решения новых классов прикладных задач. Тем самым, данные системы не предоставляют инструментальных возможностей, необходимых для их дальнейшего функционального развития.

Вместе с тем, потребность в подобных инструментах велика, поскольку разрабатывается большое число специализированных систем, ориентированных на частные индустриальные приложения. Для примера приведем лишь некоторые их них, акцентируя внимание на разнообразие прикладных постановок [9].

Система PLANETS (PLanning Activities on NETworkS) [10, 11], разработанная Университетом Каталония (University of Catalonia) в Барселоне для испанской электрической компании, как инструмент календарного планирования перестройки и технического обслуживания электрической сети без нарушения обслуживания потребителей. Система ATLAS [12, 13] осуществляет планирование производства гербицидов на заводе Monsanto в Антверпене. Система MOSES [14] была разработана компанией COSYTEC для производителя питания для животных в Великобритании. Система FORWARDC [15] представляет собой систему поддержки принятия решений (СППР) и используется на нефтеперегонных заводах в Европе при планировании поставок сырой нефти, ее обработки, смешивания и доставки потребителям. Xerox использовал систему планирования различных видов работ на копировальных машинах [16]. TAP-AI [17] — активная система планирования, предназначенная для ежедневного управления деятельностью авиалиний SAS. OPTISERVICE [18] — программный пакет для назначения персонала для всех заграничных теле- и радиостанций сети RFO с учетом ограничений по времени и условиям оплаты квалифицированных журналистов и технических работников. Система MOSAR [19], разработанная компаниями Cisi и COSYTEC для министерства юстиции Франции, назначает охранников тюрем по 200 тюрьмам Франции по чередующимся сменам. Система COBRA [20] позволяет разработать диаграммы рабочих планов машинистов поездов компании North Western Trains в Великобритании. Проект DAYSY Esprit (пакет SAS-Pilot) [17] осуществляет переназначение летных экипажей по полетам. Система краткосрочного планирования (The Short Term Planning (STP)) для компании Renault [21] решает задачу транспортировки автомобилей заказчикам с учетом множества ограничений. Средства финансового планирования применяются при утверждении бюджетов районов Москвы и их последующем контроле [22]. Планирование некоторых телекоммуникационных сетей мобильной связи осуществляется с помощью системы POPULAR [23]. Примечательно, что во многих системах используются технологии логического программирования в ограничениях, которые оказываются конкурентоспособными как по гибкости задания условий прикладных задач, так и по эффективности их решения. В частности, языки и системы логического программирования в ограничениях CHIP, 2LP, ILOG, ECLiPSe послужили математической основой для реализации некоторых из перечисленных выше специализированных систем. Тем не менее, интерпретация обобщенных задач проектного планирования в терминах логического программирования невозможна и данные технологии

могут применяться лишь в качестве элемента более общих подходов к планированию.

Важный шаг к систематизации и концептуализации задач проектного планирования был сделан в связи со становлением технологий информационного моделирования процессов строительства BIM (Building Information Modeling) [24, 25]. Появившиеся информационные стандарты и, в частности, модель IFC (ISO 16739:2013 Industry Foundation Classes) [26] позволяют специфицировать некоторые типовые условия задач календарно-сетевого планирования, используя программные интерфейсы доступа к данным или альтернативные способы их представления в файлах открытых форматов. Однако данные стандарты не регламентируют математические методы планирования и поэтому не могут служить полноценной основой для реализации программных приложений соответствующей функциональности.

Настоящая статья адресована проблемам создания инструментальной среды для программной реализации моделей, методов и приложений теории расписаний в виде объектно-ориентированного каркаса или архитектурного шаблона (object-oriented framework). В дальнейшем объектно-ориентированный каркас для приложений теории расписаний и проектного планирования называется SAF-каркас (Scheduling Application Framework). Объектно-ориентированные каркасы с успехом применяются при разработке сложных программных систем с расширяемым функционалом и при создании линеек программных продуктов в смежных предметных областях. Поэтому использование данного подхода для достижения декларируемых целей представляется вполне оправданным.

Требования, предъявляемые к объектно-ориентированному каркасу, а также общие принципы его построения обсуждаются в разделе 2. Раздел 3 посвящен организации классов прикладных данных для представления условий задач проектного планирования RCPSP в расширенных постановках. Классы математических объектов и вычислительных алгоритмов для редукции задач проектного планирования к задачам условной оптимизации и их решения описываются в разделе 4. Методологические аспекты разработки программных приложений теории расписаний на основе модулей каркаса рассматриваются в разделе 5. Результаты апробации разработанного каркаса в ходе разработки информационной системы планирования и управления проектами кратко обсуждаются в разделе 6. В заключении подводятся основные итоги работы.

2. Общие принципы и организация SAF-каркаса

2.1 Понятие объектно-ориентированного каркаса

Согласно наиболее распространённому определению [27], каркас (или фреймворк — от английского «framework») представляет собой программную платформу, определяющую структуру целевой программной

системы и облегчающую разработку, сопровождение и объединение компонентов в ее составе. Целевые системы в этом случае состоят из самого каркаса, являющегося их неизменной частью, и модулей расширения, конфигурации которых могут гибко меняться для обеспечения требуемой функциональности системы и желаемых характеристик. Организация каркаса при этом должна предусматривать, так называемые, точки расширения (hot spots), благодаря которым модули с одними и теми же интерфейсами могут применяться в качестве альтернативных реализаций основных функций системы. Точки расширения во многом задают правила конфигурирования и направления возможной функциональной эволюции целевой системы.

Каркасный подход естественным образом воплощается в рамках парадигмы объектно-ориентированного программирования (ООП). Поскольку ООП предполагает систематизацию и концептуализацию предметной области, относительно просто определяются элементы и точки расширения каркаса. Ключевые понятия предметной области, допускающие специализацию, оформляются в виде абстрактных классов каркаса с предопределенным интерфейсом. Они задают потенциальные точки расширения каркаса. Непосредственная реализация методов интерфейса осуществляется в наследуемых классах, чем обеспечивается полиморфизм включения и возможность конфигурирования приложений из специализированных элементов каркаса. Примечательно, что значительная часть методов может быть имплементирована на уровне абстрактных классов, тем самым, избавляя разработчиков от необходимости повторной реализации общих и, часто нетривиальных, механизмов взаимодействия классов каркаса. В этом случае разработчики могут сосредоточиться на особенностях реализации методов для конкретных типов классов с учетом их специфических свойств и поведения. Естественно, чтобы достичь подобных преимуществ, требуется провести тщательный объектный анализ предметной области и спроектировать каркас рациональным образом, обеспечивающим его последующее многократное использование при относительно невысоких затратах на доработку целевых приложений.

Каркасный подход к построению приложений тесто связан с общими методологиями объектно-ориентированного программирования и, в частности, с теорией объектно-компонентного моделирования, предложенной и развитой Е.М. Лаврищевой [28], а также методом построения систем с расширяемым функционалом, описанным М.М. Горбуновым-Посадовым [29].

Во многом становление каркасного подхода было предопределено необходимостью разработки графических интерфейсов пользователя (GUI), которые имели тенденцию к выделению стандартной структуры приложения и типовых графических элементов. Популярные в настоящее время

графические библиотеки MFC, Qt, GNOME, KDE в полной мере служат примерами успешного применения каркасного подхода.

Принципиальным отличием каркаса от библиотеки программ является то, что он не просто предоставляет наборы отдельных, часто несвязанных между собой функций, но и во многом предопределяет архитектуру всей целевой системы. Иногда указывают и на другое отличие, состоящее в инверсии управления и вызове пользовательских функций непосредственно из модулей каркаса [30]. Однако это наблюдение является не совсем верным. Например, в библиотеках условной нелинейной оптимизации подобная инверсия применяется с целью задания математических функций в виде соответствующих императивных процедур расчета их значений и производных. Такой способ исключает необходимость интерпретации математических функций, заданных декларативным образом, и повышает эффективность вычислительного процесса.

В любом случае организация каркаса не препятствует интеграции любого числа сторонних библиотек самой разной функциональности для решения вспомогательных задач. Более того, она может предусматривать применение специализированных языков для описания прикладных задач, форматов обменных файлов для вывода и хранения результатов, языков запросов к СУБД, протоколов взаимодействия клиентских и серверных приложений и т.п. Тем самым, концепция каркаса существенно расширяет идею библиотечной организации программного обеспечения, предусматривая развитые инструментальные возможности для построения целевых приложений.

2.2 Общие требования и принципы построения SAF-каркаса

Обсуждаемый объектно-ориентированный SAF-каркас сочетает в себе функции математической библиотеки и инструментальной среды для построения программных приложений теории расписаний и проектного планирования. Сформулируем общие требования, которые предъявлялись к нему и учитывались при его разработке:

- универсальность, предполагающая наличие готовых к использованию программных модулей для математически строгой постановки и решения типовых задач теории расписаний и проектного планирования;
- эффективность, означающая в данном случае равномерно высокую производительность модулей для решения обсуждаемого класса задач и, прежде всего, для приближенного решения индустриально значимых задач проектного планирования высокой размерности;
- гибкость, подразумевающая возможность повторного использования имеющихся модулей при программной реализации новых моделей, методов и приложений теории расписаний при относительно низких затратах на доработку.

Данные отчасти противоречивые требования нуждаются в уточнениях, поскольку в значительной степени формируют общий функциональный облик всего каркаса.

Как известно, теория расписаний охватывает довольно много классов задач с разными оценками вычислительной сложности и со своими алгоритмами решения. Общепринятая нотация Грэхема $\alpha|\beta|\gamma$, в которой характеристики описывают соответствующие модели исполнения операций и машин (работ и ресурсов) и целевые функции, задает общую классификацию подобных задач. В зависимости от индивидуальных характеристик $\alpha|\beta|\gamma$ вычислительная сложность составления расписания может существенно варьироваться и поэтому для этих целей обычно применяют специальные алгоритмы, ориентированные на частные классы задач.

Важно отметить, что при разработке каркаса, как универсальной математической библиотеки, не ставилась цель предоставить средства, которые бы обеспечили решение всех задач теории расписания за оптимальное время. Вместо этого предпринята попытка эффективно решать задачи проектного планирования в постановке RCPSP, к которой редуцируются все основные задачи теории расписаний [31]. Однако и данная классическая постановка оказывается довольно частной для реализации приложений календарно-сетевого планирования и управления проектами, в которых применяются сложные многопараметрические модели работ, связей, ресурсов, календарей, счетов. Некоторые отличия в постановках задач указаны в работе [1], в которой авторы говорят о задаче RCPSP в расширенной постановке. Необходимые математические обобщения также обсуждаются и систематизируются в нашей обзорной работе [32].

В работе [31] определяется класс задач обобщенного проектного планирования Generally Constrained Project Scheduling Problem (GCPSP) и формулируются утверждения о сводимости задач RCPSP в расширенных постановках к задачам GCPSP. Существенно, что последние формулируются в математически нейтральной форме, которая определяет лишь тип целевой функции и вид алгебраических ограничений, возникающих в задачах проектного планирования. Таким образом, универсальность каркаса может обеспечиваться путем предоставления развитого набора программных модулей для задания условий задач проектного планирования в постановке GCPSP и их решения. Для математической редукции прикладных задач достаточно проинтерпретировать их условия в терминах постановки проектного планирования GCPSP, разрешить ее и представить результаты в представлении исходной задачи.

Требования эффективности также нуждаются в некоторых пояснениях. Поскольку задачи проектного планирования RCPSP и GCPSP являются NP-полными, а для индустриальной практики представляют интерес проекты с количеством работ, исчисляемых десятками и сотнями тысяч, главное внимание должно уделяться быстрым алгоритмам, обеспечивающим поиск

приближенных решений за полиномиальное время. Лучший из известных точных алгоритмов Брукера за приемлемое время может решать задачи размерности не больше 60 [33], что делает невозможным его использование в обсуждаемых индустриальных приложениях. Вместе с тем, точные алгоритмы могут применяться для валидации приближенных алгоритмов и, в частности, для выбора и настройки применяемых в них эвристических правил. Поэтому состав каркаса может предусматривать программные модули, реализующие и некоторые точные алгоритмы. Однако вопросы эффективности становятся некритичными, поскольку оценка качества найденных приближенных решений может осуществляться на тестовых задачах очень низкой размерности.

Под гибкостью каркаса как программно-инструментальной среды подразумевается возможность реализации новых математических моделей, методов и приложений теории расписаний при относительно низких затратах на доработку имеющихся программных модулей. Поскольку задачи теории расписаний редуцируются к соответствующей обобщенной постановке проектного планирования GCPSP, для задания условий и решения которой основные программные модули уже реализованы и включены в состав каркаса, доработка целевых приложений потребует небольших затрат. В тех случаях, когда необходимо решать частные классы задач за оптимальное время, потребуются дополнительные усилия на программную реализацию специальных алгоритмов. В предположении, что они основаны на уже реализованных в каркасе алгоритмах или используют общую с ними вычислительную стратегию, подобные затраты также могут быть минимизированы.

Таким образом, обсуждаемые требования универсальности, эффективности и гибкости, предъявляемые к каркасу, могут быть удовлетворены на основе изложенных выше принципов.

2.3 Организация и состав классов SAF-каркаса

Разработанный каркас представляет собой систему классов (в дальнейшем, учитывая практическую реализацию на языке Си++, будем использовать принятые термины «класс», «конкретный класс», «абстрактный класс» и «интерфейс»). В организации каркаса выделим следующие группы классов:

- классы решателей (Solvers), реализующие общие алгоритмические схемы решения задач GCPSP (Schedulers), а также эвристики для поиска приближенных решений (Heuristics);
- классы математических объектов (Mathematics), предназначенные для задания условий и представления результатов проектного планирования в обобщенной постановке GCPSP;
- классы математической редукции (Reductions), обеспечивающие сводимость прикладных задач составления расписаний к постановке

GCPSP и соответствующую интерпретацию прикладных данных;

- классы прикладных данных (Applications), используемые для представления условий и результатов решения задач проектного планирования RCPSP в расширенных постановках;
- классы средств визуализации (Visualizations), предназначенные для графического отображения результатов планирования, в том числе, с использованием текстовых отчетов, графиков, диаграмм.

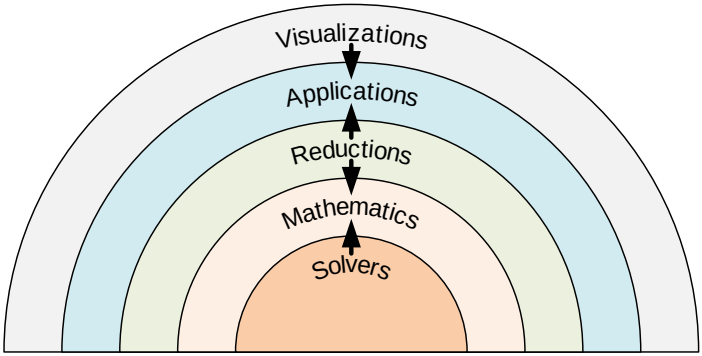

Рис. 1. Организация пакетов классов каркаса и основные отношения использования
Fig. 1. The organization of packages of classes of the framework and the basic relations of use

Классы средств визуализации будут рассмотрены в заключительных разделах, посвященных методологии разработки приложений, внедрению разработанного каркаса и созданию перспективной системы визуального планирования и управления проектами.

В следующих разделах остановимся более подробно на основных группах классов, непосредственно связанных с решением задач теории расписаний. Заметим, что часть из них реализуется как конкретные классы, допускающие непосредственное конструирование объектов. Другая часть представляет собой интерфейсы или абстрактные классы, которые по существу определяют точки расширения каркаса и позволяют предоставить их альтернативные реализации. Примечательно, что некоторые функции каркаса, в частности общие алгоритмические схемы, могут реализовываться в абстрактных классах без конкретизации типов условий решаемых задач и особенностей конкретных алгоритмов. Более того, такой способ реализации каркаса является рациональным с точки зрения повторного использования модулей и функциональной эволюции целевых приложений.

3. Организация классов прикладных данных

В данном разделе подробно описываются классы и интерфейсы каркаса, относящиеся к группе Applications и позволяющие задать условия и

результаты задач проектного планирования в расширенных постановках RCPSP.

3.1 Класс «Проект» (Project)

Класс Project агрегирует в себе все данные, необходимые для математически корректной постановки задачи проектного планирования, и предоставляет необходимые интерфейсы доступа к ним. Сам проект представляется иерархией связанных между собой работ с назначенными календарями, ресурсами и счетами. В рамках ООП перечисленные понятия реализуются соответствующими классами Project, Task, Link, Calendar, Resource, Account соответственно. Классы Task и Resource являются абстрактными, что означает невозможность создания экземпляров и необходимость предоставления конкретных реализаций методов, объявленных в интерфейсах данных классов. В следующих подразделах подробно описываются особенности подобных реализаций. В частности, поясняются способы определения простых и составных типов работ и ресурсов. Дополнительные классы TaskRate, ResourceRate, ResourceUse, Supply и Replenishment используются для ассоциирования работ, ресурсов и счетов между собой и параметризации подобных отношений.

Перечисленные выше классы являются конкретными, однако следует принять во внимание, что в их основу положены довольно общие параметрические модели, охватывающие расширенные постановки обсуждаемого класса задач RCPSP. Вместе с тем, при необходимости данные модели могут быть развиты и реализованы путем непосредственного наследования классов каркаса.

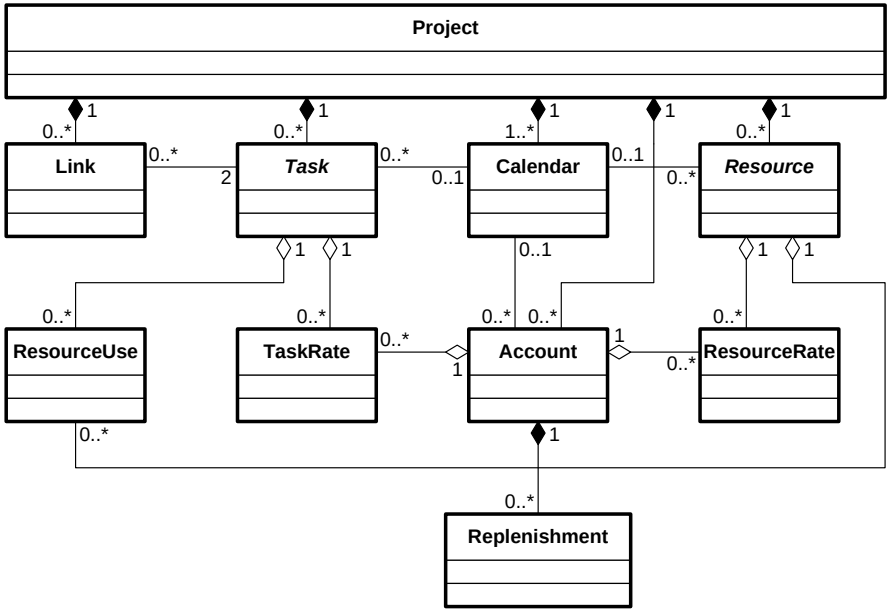

Рис. 2. UML-диаграмма основных классов прикладных данных
Fig. 2. UML diagram of main classes of application data

Кроме агрегации экземпляров указанных на диаграмме классов, класс Project определяет собственные атрибуты. Такими атрибутами являются проектный календарь, используемый в качестве основного в тех случаях, когда не определён индивидуальный календарь для работ, ресурсов и счетов, а также временные проектные ограничения, определяющие время начала и/или завершения проекта и возможную стратегию прямого и обратного планирования проектных работ (как можно раньше и как можно позже соответственно).

3.2 Календарные данные

Работа с календарной информацией занимает важное место при постановке и решении задач проектного планирования. Сами календари представляют собой объекты с довольно сложной организацией данных и нетривиальными операциями пересчета календарных дат, времен, рабочих интервалов. Поэтому в их реализации используются вспомогательные классы Time, Date, DateAndTime, TimeInterval, Duration, DayOfWeek, WorkWeek, MonthOfYear, RecurrencePattern, RecurrenceTimeInterval, которые в составе каркаса выполняют и самостоятельные функции. Рассмотрим их более подробно.

3.2.1 Класс «Время» (Time)

Класс Time предназначен для представления времени в рамках одних суток с точностью до долей секунды. Переменная времени может принимать любое значение от 00:00:00 до 24:00:00. Класс реализует методы для установки времени суток с помощью компонентов-значений часа, минуты, секунды и долей секунды, а также методы для получения соответствующих компонентов-значений времени суток и его строкового представления. Интерфейс класса предусматривает необходимые логические операции сравнения текущего времени с заданным значением, а также арифметические операции добавления заданной продолжительности к текущему времени и ее вычитания из текущего времени. Результат последних операций всегда приводится к суточному временному интервалу.

3.2.2 Класс «Дата» (Date)

Класс Date используется для представления календарной даты в виде компонентов-значений числа, месяца и года. Интерфейс класса предусматривает методы для установки календарных дат, сравнения дат, добавления к дате и вычитания из неё заданной продолжительности, вычисления продолжительности временного интервала между текущей датой и заданной.

3.2.3 Класс «Дата и время» (DateAndTime)

Класс DateAndTime предназначен для консолидированного представления календарной даты и времени суток, позволяющего оперировать абсолютными временными метками. Интерфейс класса во многом повторяет интерфейсы рассмотренных выше классов Date и Time.

3.2.4 Класс «Временной интервал» (TimeInterval)

Класс TimeInterval позволяет оперировать временными интервалами в пределах одних суток. Временной интервал задается нижней и верхней границей в предположении, что нижняя граница принадлежит интервалу, а верхняя — нет. Класс предоставляет методы для задания и получения границ временного интервала, вычисления его продолжительности, определения статуса принадлежности заданного момента времени текущему интервалу, а также вычисления теоретико-множественных операций пересечения и объединения текущего интервала с заданным интервалом.

3.2.5 Класс «Продолжительность» (Duration)

Данный класс используется для представления продолжительности проектных работ и лагов между ними. Продолжительность исчисляется с точностью до долей секунды и может быть положительной, отрицательной или нулевой. В классе реализуются арифметические операции сложения и вычитания продолжительностей, операции умножения и деления текущей продолжительности на число, а также логические операции сравнения заданных продолжительностей.

3.2.6 Тип данных «День недели» (DayOfWeek)

Тип данных DayOfWeek предназначен для представления дней недели и естественным образом реализуется как перечислимый тип с семью предопределенными значениями: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday и Sunday.

3.2.7 Тип данных «Месяц года» (MonthOfYear)

Тип данных MonthOfYear предназначен для представления месяцев в году и реализуется в каркасе как перечислимый тип с двенадцатью предопределенными значениями: January, February, March, April, May, June, July, August, September, October, November и December.

3.2.8 Класс «Рабочая неделя» (WorkWeek)

Класс WorkWeek позволяет приписать логические признаки дням недели, например, с целью задания и определения их рабочего статуса. Класс реализуется как массив из семи логических значений, каждое из которых соответствует определенному дню недели в перечислении DayOfWeek. Интерфейс класса позволяет определить статус заданного дня недели и при необходимости изменить его.

3.2.9 Тип данных «Регулярное правило» (RecurrencePattern)

Обычно временные интервалы в рабочих календарях подчиняются некоторым регулярным правилам. Например, рабочие часы организации могут повторяться «ежедневно», «еженедельно», «ежемесячно», «в определённое число каждого месяца» и т. д. Для описания подобных регулярных правил в составе каркаса предусмотрен вспомогательный перечислимый тип данных RecurrencePattern со следующими предопределенными значениями: Daily, Weekly, Monthly, MonthlyByDayOfMonth, MonthlyByPosition, ByDayCount, ByWeekdayCount, YearlyByDayOfMonth, YearlyByPosition. Именованные значения имеют понятную интерпретацию.

3.2.10 Класс «Регулярный временной интервал» (RecurrenceTimeInterval)

Класс RecurrenceTimeInterval предназначен для задания регулярных временных интервалов. Отличием от рассмотренного выше класса TimeInterval является более компактный, не избыточный способ представления временных интервалов в тех случаях, когда они подчиняются регулярным правилам, специфицируемым типом RecurrencePattern. Использование в подобных случаях класса TimeInterval привело бы к необходимости конструирования и последующего анализа огромного числа интервальных объектов для каждого проектного дня. Класс RecurrenceTimeInterval обобщает модель данных класса TimeInterval путем определения дополнительных атрибутов для задания регулярного правила, начальной и конечной даты его применения и соответствующих ему параметров (например, «день недели», «число месяца», «день года» и т.п.).

3.2.11 Класс «Календарь» (Calendar)

Исполнение работ и привлечение ресурсов в рамках проектной деятельности обычно осуществляется на основе рабочих календарей. Корректная постановка задачи проектного планирования предполагает, что должен быть определен, по крайней мере, один проектный календарь, используемый по-умолчанию. Для работы с календарями в состав объектно-ориентированного каркаса включен конкретный класс Calendar.

Календари могут быть рационально организованы в виде двух множеств, одно из которых соответствует регулярным интервалам рабочего времени, а другое — их исключениям в особые календарные дни. В классе Calendar для этих целей используется две коллекции workTime и exception с элементами соответствующего типа RecurrenceTimeInterval. Первая коллекция используется для описания типовой рабочей недели (например, с понедельника по пятницу с 9:00 до 17:00). Вторая коллекция используется для описания исключений, которые могут происходить, например, в праздничные или предпраздничные дни. Фактическое рабочее расписание получается путем теоретико-множественной операции вычитания исключительных интервалов из основных рабочих интервалов. Выполнение данной операции может быть сопряжено со значительным объемом вычислений, поэтому следует избегать избыточной фрагментации исключений. Например, ежедневный обеденный перерыв с 13:00 до 14:00 нерационально описывать в виде исключения, а лучше представить прерываемыми рабочими интервалами.

Календари организуются в виде иерархической структуры с введенным отношением наследования между родительскими и дочерними элементами. Наследуемый календарь может уточнять или переопределять фактическое рабочее расписание родительского календаря. Предполагается, что собственные рабочие интервалы расширяют расписание родителя, а собственные исключения — ограничивают. Фактическое расписание наследника получается путем объединения собственных рабочих интервалов с рабочими интервалами родителя после вычитания из них исключительных интервалов родителя и последующего вычитания из полученного результата исключительных интервалов наследника.

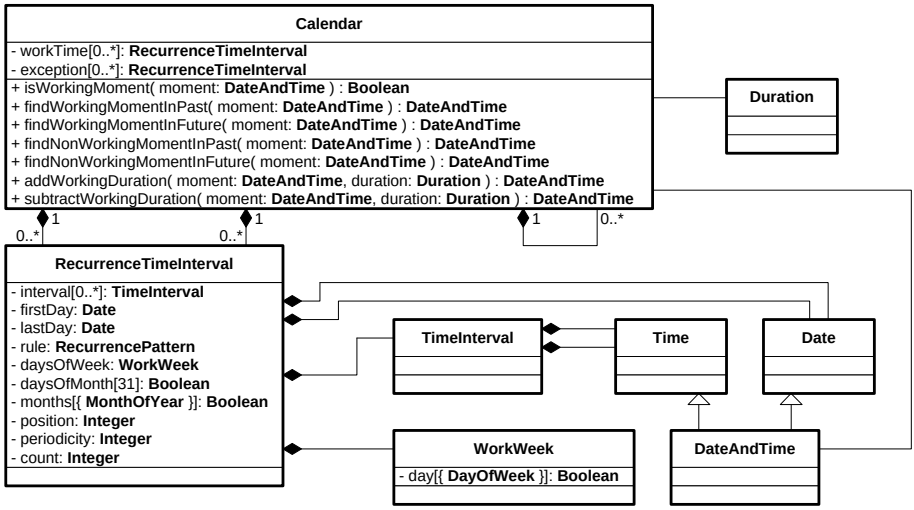

Рис. 3. UML-диаграмма классов календарных данных
Fig. 3. UML diagram of calendar data classes

Класс Calendar реализует развитый набор операций для определения статуса заданной даты или заданного временного интервала, вычисления ближайших рабочих и нерабочих дат календаря, а также для пересчета даты завершения (или начала) работы по заданной дате ее начала (или завершения) и продолжительности. Интерфейс класса предусматривает также операции unite и intersect, которые позволяют сконструировать новые календари, фактическое расписание которых является объединением или пересечением расписаний заданных календарей-операндов. Данные операции упрощают реализацию основных вычислительных процедур составления расписаний с учетом календарей, которые могут быть индивидуально приписаны проектным работам, ресурсам, ограничениям предшествования и нуждаются в комплексном анализе при пересчете фактических дат начала и завершения проектных работ.

3.3 Проектные работы

Каркас использует естественное разделение проектных работ на простые и составные. Простыми работами являются элементарные активности Activity, вехи StartMilestone, FinishMilestone и «гамаки» ShortHammock, LongHammock. Для многоуровневого представления проектного плана в виде иерархии работ используются структуры работ WBS (принятое сокращение от Work Breakdown Structure) и мультимодальные работы MultimodalTask. Все перечисленные виды работ в каркасе реализуются соответствующими конкретными классами, наследуемыми от общего абстрактного класса Task.

Мы допускаем, что некоторые атрибуты, декларируемые в нем, могут оказаться производными в конкретных реализациях и поэтому попытки их установки могут приводить к исключительным ситуациям. Тем не менее, наличие общего интерфейса Task оказывается важным фактором, предопределяющим единую дисциплину реализации классов работ.

3.3.1 Класс «Работа» (Task)

Абстрактный класс Task определяет общие методы получения и задания общих параметров работы. Прежде всего, это — планируемые и фактические даты начала и завершения работы, ее продолжительность, рабочий календарь, наложенные ограничения, используемые ресурсы, стоимость, счёт, приоритет, режим исполнения, статус и процент выполнения. В зависимости от вида работы логика реализации методов претерпевает существенные изменения и поэтому вынесена на уровень конкретных классов. Особенности полиморфной реализации далее обсуждаются на примере методов получения временных параметров работы.

Кратко уточним назначение перечисленных атрибутов работ. Рабочий календарь — календарь, в соответствии с которым осуществляется планирование и исполнение работы. В определении класса Task он реализуется опциональной ссылкой на объекты рассмотренного выше класса Calendar. При ее отсутствии применяется календарь проекта, наличие которого обязательно при конструировании объектов класса Project.

Статус определяет планируемое, стартованное, прерванное, возобновленное или финишированное состояние работы и представляется перечислимым типом TaskStatus с соответствующими значениями Planned, Started, Suspended, Resumed, Finished. Приоритет — натуральное число, характеризующее предпочтения пользователя по приоритизации работ в ходе составления расписания. Обычно необходимость в их использовании возникает, когда одновременное выполнение работ невозможно из-за ограниченности общих ресурсов и требуется принять решение о порядке их выполнения.

Явные временные ограничения определяют допустимые интервалы или полуинтервалы для дат начала или завершения работ. Данные ограничения снабжаются спецификатором обязательности или приоритетности над ограничениями предшествования в тех случаях, когда формируемая система алгебраических уравнений и неравенств перегружена и не может быть полностью разрешена. Если спецификатор предписывает обязательное выполнение временного ограничения и это препятствует каким-либо ограничениям предшествования, то расписание строится до конца, но снабжается отчетом о неразрешенных ограничениях.

Правило выравнивания устанавливает необходимость принудительного выравнивания планируемой даты на начало или конец каждой минуты, часа, суток, недели, месяца или года и представляется в каркасе перечислимым типом SnapMode с соответствующими значениями SnapToMinute,

SnapToHour, SnapToDay, SnapToWeek, SnapToMonth, SnapToYear. Правило выравнивания является опциональным атрибутом работы и может рассматриваться в качестве специфического временного ограничения, определяющего область допустимых значений для даты начала работы.

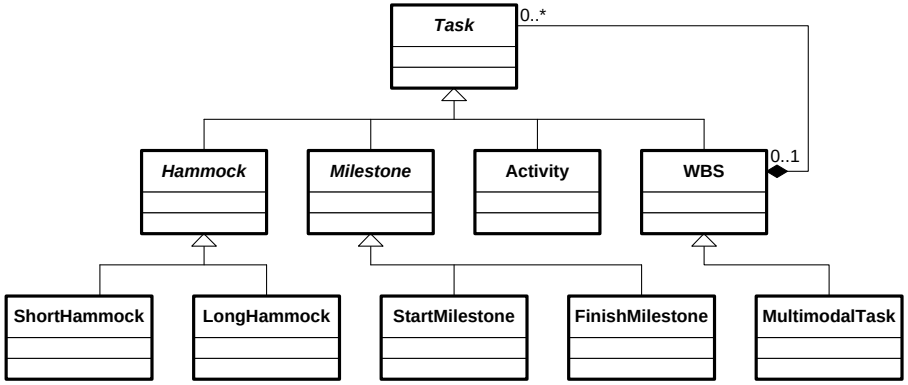

Рис. 4. UML-диаграмма иерархии классов работ
Fig. 4. UML diagram of the hierarchy of task classes

Для некоторых видов работ может быть указано максимально допустимое число прерываний, которое определяет возможные режимы их исполнения. Если данный атрибут не задан, то предполагается, что количество прерываний не ограничено. Нулевое значение интерпретируется как недопустимость прерываний.

3.3.2 Класс «Активность» (Activity)

Класс Activity реализует понятие простых операций, которые представляют собой терминальные работы в иерархическом представлении проектного плана. Конкретный класс Activity наследует и реализует интерфейс класса Task таким образом, что любой параметр работы может быть задан индивидуально, но при одновременной коррекции других связанных с ним параметров. Тем самым, обеспечивается семантическая согласованность представления данных. Например, при установке планируемых дат начала и завершения работы, пересчитывается ее продолжительность. При установке доли выполнения пересчитывается прогнозируемое время завершения работы и т.п.

3.3.3 Классы «Вехи» (Milestone)

Близкое поведение реализуют классы StartMilestone и FinishMilestone. Основным отличием вех от простых операций является нулевая продолжительность и, как следствие, совпадающие даты начала и завершения работ. При этом даты в классе StartMilestone принудительно выравниваются на момент времени, допустимый для начала работы, а в классе FinishMilestone

— на момент времени, допустимый для завершения работы. Установка даты начала вехи приводит к коррекции даты завершения и наоборот. Вызов метода установки продолжительности для вехи порождает соответствующее исключение.

3.3.4 Класс «Структура работы» (WBS)

Главной особенностью реализации класса WBS является наличие множественной композиции children на объекты типа Task, благодаря которой структуры работ могут содержать в себе дочерние работы любых типов, в том числе и работы более низких уровней. Таким образом, класс WBS позволяет структурировать проектный план в виде многоуровневой иерархии разнотипных работ. Структуры WBS не обязаны содержать дочерних работ, поскольку детализация проектного плана обычно происходит постепенно.

Явное задание временных параметров и других производных атрибутов WBS, используя методы наследуемого интерфейса Task, является некорректным и приводит к исключительным ситуациям. В самом деле, временные, ресурсные и стоимостные характеристики структуры работ определяются ее дочерними работами. Например, дата начала структуры работы определяется самым ранним стартом дочерних работ, а дата ее завершения — самым поздним финишем. Вычисление данных параметров осуществляется путем рекурсивного обхода многоуровневого представления структуры работ и уточнения минимальных и максимальных значений соответствующих дат в дочерних работах. Аналогичным образом вычисляются ресурсные и стоимостные характеристики структур работ. Последние, в частности, применяются при оценке качества найденных решений в постановках, нацеленных на минимизацию сроков и стоимости проекта в условиях жестких ресурсных ограничений.

3.3.5 Классы «Гамаки» (Hammock)

Подобно структурам работ WBS реализуются «гамаки» классов ShortHammock и LongHammock. Они не предусматривают явного задания дат работ, поскольку рассчитываются по временным параметрам предшествующих и последующих работ. Дата начала «Короткого гамака» определяется самым поздним финишем из всех предшественников, а дата его завершения — самым ранним стартом последователей. Для «Длинного гамака» дата начала совпадает с самым ранним финишем из всех предшественников, а дата завершения — с самым поздним стартом последователей. В случае отсутствия предшественников и/или последователей «гамак» вырождается в работу с продолжительностью, равной продолжительности проекта.

3.3.6 Класс «Мультимодальная работа» (MultimodalTask)

Важным требованиям к средствам планирования сложных проектов является возможность задания альтернативных режимов выполнения. С этой целью в состав каркаса включен класс MultimodalTask, определение которого во многом повторяет класс WBS из-за использования композиции дочерних

работ children. Однако логика реализации интерфейса Task принципиально отличается, поскольку выполнение структуры работ означает выполнение всех дочерних работ, а выполнение мультимодальной работы предполагает исполнение лишь одной из дочерних работ. Поскольку тип дочерних работ не конкретизируется композицией children в WBS и MultimodalTask, с их помощью удается строить сложные стратегии проектной деятельности, например, многоуровневые альтернативы структур работ. Однако следует иметь в виду, что из-за комбинаторной неопределенности наличие мультимодальных работ в представлении проекта существенно усложняет поиск расписаний, близких к оптимальным.

3.4 Класс «Связь работ» (Link)

Класс Link предназначен для задания отношений предшествования и синхронизации между работами при постановке задач проектного планирования. В классе определяются две обязательные ассоциации на объекты типа Task, одна из которых указывает на предшествующую работу (upstreamTask), а другая — на последующую (downstreamTask). Четыре опциональных атрибута типа Duration определяют минимальную и максимальную задержку синхронизации, пересчитанную в календарные даты с использованием рабочих календарей предшественника и последователя. Незаданные минимальные задержки (upstreamMinLag и downstreamMinLag) интерпретируются как нулевые, а незаданные максимальные задержки (upstreamMaxLag и downstreamMaxLag) — как бесконечно большие величины. Опциональная ссылка на календарь типа Calendar используется для пересчета дат в соответствии с собственным календарем.

Кроме этого, объекты класса Link имеют в качестве атрибута спецификатор связи, представленный перечислимым типом LinkType. Данный атрибут может принимать одно из следующих значений: SSLink, SFLink, FSLink или FFLink. Значение SSLink означает, что связь устанавливается между началом предшественника и началом последователя, SFLink — между началом предшественника и окончанием последователя, FSLink — между окончанием предшественника и началом последователя и FFLink — между окончанием предшественника и окончанием последователя. С учётом данного спецификатора минимальные и максимальные задержки приобретают более понятный смысл. Например, для связи с типом FSLink при заданной минимальной задержке данный вид связи устанавливает требование начать последующую работу не раньше, чем через установленное время после завершения предшествующей работы. При заданной максимальной задержке — не позже, чем через установленное время после завершения предшествующей. Примечательно, что задержки могут быть отрицательными и приводить к обратной последовательности выполнения предшественников и последователей.

3.5 Ресурсы

Каркас поддерживает несколько категорий ресурсов, реализуемых соответствующими классами простого ресурса SimpleResource, группового ресурса GroupResource, объединения ресурсов JointResource и семейства ресурсов FamilyResource. За исключением простого ресурса все остальные виды являются составными, что предполагает композицию дочерних или ассоциацию сторонних ресурсов.

3.5.1 Класс «Ресурс» (Resource)

Абстрактный класс Resource определяет базовый тип, от которого наследуются все перечисленные выше конкретные классы. Данный класс не предусматривает общий интерфейс для получения и задания ресурсных параметров в силу того, что простые и составные ресурсы параметризуются различными способами. Единственными общими методами, определяемыми в данном классе, являются правила исчисления стоимости ресурса.

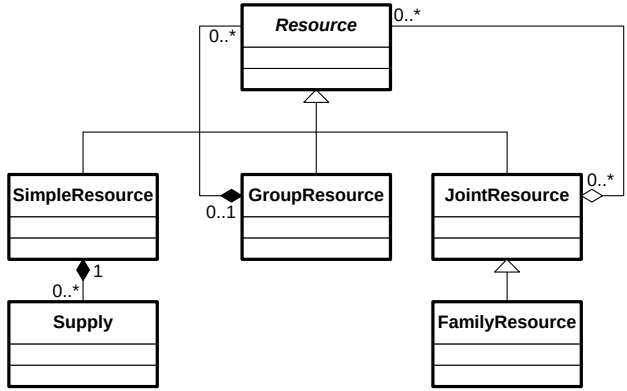

Рис. 5. UML-диаграмма иерархии классов ресурсов
Fig. 5. UML diagram of the hierarchy of resource classes

3.5.2 Класс «Простой ресурс» (SimpleResource)

Класс SimpleResource реализует понятие простого ресурса со следующим набором параметров. Это — признаки возобновимости и разделяемости ресурса, рабочие календари, временные лаги и доступное количество ресурса.

Признак возобновимости представляется перечислимым типом со значениями Renewable и Expendable. Первое значение указывает на возврат используемого количества ресурса по окончании каждой работы в общий пул. Второе устанавливает, что ресурс тратится в ходе выполнения каждой работы, где он используется, и его доступное количество в ходе выполнения проекта только уменьшается. Обычно к возобновимым ресурсам относят рабочий персонал и технику, а к невозобновимым — расходные материалы и энергетические ресурсы.

Признак разделяемости представляется перечислимым типом со значениями Discrete и Continuous, устанавливающими привлекается ли ресурс дискретным образом или непрерывным. Например, если некоторая работа выполняется за один день и ее трудоемкость составляет 1,5 человеко-дня, то данный признак позволяет решить, необходимо ли привлечь для ее выполнения двух сотрудников на целый день (Discrete) или в условиях неполной занятости они могут одновременно участвовать в других параллельных работах (Continuous). Следует отметить, что доступное количество дискретных ресурсов всегда исчисляется в целых. Оба признака являются обязательными атрибутами простого ресурса, поскольку участвуют в оценке его доступности и влияют на логику составления расписания.

Основной календарь устанавливает рабочее время, в которое ресурс доступен для проектных работ. Обычно в качестве основного календаря используется проектный календарь, в соответствии с которым выполняются все основные работы. Однако возможны ситуации, когда ресурсные календари могут иметь отличия, например, в силу технологических особенностей применяемого оборудования или индивидуальных планов сотрудников. В подобных случаях, фактический календарь выполнения работы строится путем теоретико-множественного пересечения ее собственного календаря с календарями всех используемых ресурсов. Для поддержки основного календаря используется обязательная объектная ссылка на класс Calendar.

Дополнительный календарь реализуется аналогичным образом, но является необязательной объектной ссылкой. Его основное назначение — планирование работ в сверхурочное время, обычно оплачиваемое по повышенным тарифам. Если целью планирования является скорейшее завершение проекта за счет возможного увеличения бюджета, то расписание строится с учетом основного и дополнительного календарей.

Временные лаги ресурса определяются как два опциональных атрибута класса, имеющие тип продолжительности Duration. Если атрибуты не установлены, то они интерпретируются как имеющие нулевую продолжительность. Первый атрибут определяет время доставки, установки или наладки ресурса перед его использованием в ходе выполнения работы, а второй атрибут — время остановки, демонтажа или возврата ресурса, необходимое для его освобождения и последующего использования в других работах. Временные лаги имеют спецификатор зависимости от количества используемого ресурса. При его задании итоговый временной лаг получается домножением на количество ресурса, используемого в конкретной работе. Перечисленные временные параметры также применяются в случаях прерывания и возобновления работ аналогично тому, как это происходит при их начале и завершении.

3.5.3 Класс «Поставка» (Supply)

Для получения доступного количества простого ресурса на заданный момент времени часто необходимо реконструировать историю поставок. С это целью в состав каркаса включен специальный класс Supply, а в классе SimpleResource определяется множественная композиция объектов данного класса. Множество объектов класса Supply реализует понятие «Цепочки поставок» путем определения для каждого объекта двух наборов атрибутов, определяющих планируемые и фактические значения следующих параметров: дата поставки, объем поставки или списания ресурса, стоимость организации всей поставки, стоимость за единицу ресурса, а также счет, если он отличается он единого счета ресурса. Атрибуты, связанные с планируемыми величинами, являются обязательными, а атрибуты, связанные с фактическими величинами — опциональными, поскольку актуализация поставок осуществляется уже в процессе проектной деятельности.

Объем поставки является ее ключевой характеристикой, поскольку предопределяет количество ресурса, доступное на начало проекта или на начало любой спланированной работы. В ходе выполнения проекта ресурсы могут захватываться, потребляться, освобождаться, генерироваться, в результате чего их доступное количество меняется.

При составлении расписания важно учесть, что на протяжении всего проекта доступное количество каждого ресурса не может быть отрицательным. Нарушение этого ограничения означает, что проектные работы спланированы неправильно и используют несуществующие объемы ресурса. Если проект не предусматривает пополнение или генерацию ресурса, то ни одна работа не может использовать ресурса больше, чем доступно на начало проекта. Это условие может проверяться уже на этапе постановки задачи.

3.5.4 Класс «Групповой ресурс» (GroupResource)

Класс группового ресурса GroupResource применяется для организации разнородных ресурсов в единое иерархическое многоуровневое представление в соответствии с требованиями пользователей. С этой целью в данном классе, наследуемом от абстрактного интерфейса Resource, определяется множественная композиция composedOf объектов типа Resource. Это позволяет в каждый групповой ресурс включить любое количество разнородных ресурсов, в том числе и групповые ресурсы более низких уровней.

3.5.5 Класс «Объединение ресурсов» (JointResource)

Класс объединения ресурсов JointResource во многом аналогичен классу GroupResource за исключением того, что вместо композиции composedOf определяется множественная ассоциация assembledFrom на объекты типа Resource. Класс предназначен для задания альтернативных способов группирования разнородных ресурсов, например, при формировании и

комплектовании бригад. Назначение объединенного ресурса на работу предполагает использование в работе всех его ассоциируемых ресурсов.

3.5.6 Класс «Семейство ресурсов» (FamilyResource)

Класс семейства ресурсов FamilyResource аналогичен классу JointResource и использует множественную ассоциацию associatedWith типа Resource, что позволяет в одном объекте группировать разные ресурсы. Однако принципиальным семантическим ограничением является требование, чтобы все ассоциируемые ресурсы были однородными. Например, если один из простых ресурсов является возобновимым, то и все остальные ресурсы, включенные в семейство, должны быть возобновимыми. Семейства ресурсов непосредственно назначаются на работы, однако любое такое назначение допускает использование любых комбинаций ассоциируемых родственных ресурсов. Например, если семейство ресурсов определяет группу сотрудников соответствующей специальности и квалификации, то назначение семейства на работу будет означать, что для выполнения работы может привлекаться любой свободный сотрудник или сотрудники, незанятые в период выполнения работы.

3.5.7 Класс «Использование ресурса» (ResourceUse)

Класс ResourceUse позволяет ассоциировать работу и используемый ей ресурс путем установки ссылок на соответствующие объекты и задания значений атрибутов, определяющих условия привлечения ресурса, включая количество или лимиты использования ресурса, профиль потребления или генерации ресурса на протяжении работы.

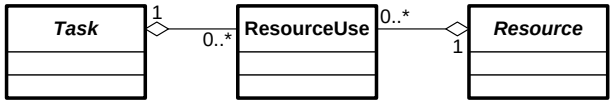

Рис. 6. UML-диаграмма класса использования ресурсов
Fig. 6. UML diagram of resource use class

Последний атрибут определяется как перечислимый тип ResourceProfile со значениями BackLoaded, BellShaped, FrontLoaded, Linear, OffsetTriangular, ThreeStep, Trapezoidal, TriangularDecrease, TriangularIncrease, DoublePeak, EarlyPeak, определяющими вид соответствующих скалярных функций одной переменной. Реальные профили потребления ресурса получаются путем масштабирования нормированных функций по оси абсцисс на период выполнения работы и по оси ординат на количество привлекаемого ресурса.

Следует иметь в виду, что установленное количество ресурса может быть отрицательным, что означает генерацию работой данного ресурса и возможность использования дополнительного количества другими работами. Примерами работ, генерирующих возобновимые и невозобновимые ресурсы,

могут служить краткосрочная аренда дополнительного оборудования и производство вспомогательных материалов в ходе проектной деятельности.

3.6 Финансовое обеспечение

Важным аспектом проектной деятельности является бюджетно-финансовое планирование и обеспечение. С точки зрения дизайна каркаса ключевыми элементами здесь являются бюджетный счет, а также различные правила исчисления стоимости работ и ресурсов.

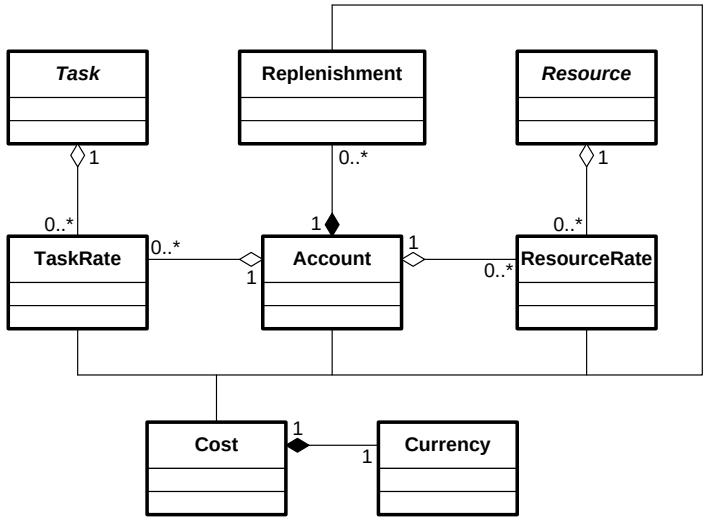

Рис. 7. UML-диаграмма классов финансового обеспечения
Fig. 7. UML diagram of financial support classes

3.6.1 Классы «Стоимость» (Cost) и тип данных «Валюта» (Currency)

Вспомогательный класс Cost реализует понятие стоимости, выраженной в денежно-валютном эквиваленте. Объекты данного класса представляются парой значений: денежным номиналом и видом валюты, в которой данный номинал представлен. Вид валюты задается перечислимым типом данных Currency. Номинал может быть положительным, отрицательным или нулевым. Для объектов класса определены арифметические операции сложения, вычитания, умножения и деления на вещественное число и логические операции сравнения. В случаях, когда валюты двух стоимостных операндов не совпадают, значения номиналов приводятся к единой валюте по предопределенному обменному курсу. Для подобных целей класс предусматривает статические методы установки кросс-курсов и применения их к заданным номиналам.

3.6.2 Тип данных «Стоимость за…» (CostType)

Тип данных CostType предназначен для спецификации выставляемой стоимости, ставки или тарифа. Другими словами, атрибуты данного типа позволяют определить, за что указана стоимость. Данный тип представлен в каркасе как перечисляемый тип со следующими предопределёнными значениями: FixedCost (стоимость фиксирована и не зависит от какой-либо продолжительности или количества), CostPerMinute, CostPerHour, CostPerDay, CostPerWeek, CostPerMonth, CostPerYear (приведенная стоимость за соответствующую единицу времени), CostPerUnitMinute, CostPerUnitHour, CostPerUnitDay, CostPerUnitWeek, CostPerUnitMonth, CostPerUnitYear (приведенная стоимость за использование одной единицы ресурса в течении единицы времени), CostPerUnit (стоимость за использование одной единицы ресурса).

3.6.3 Класс «Счёт» (Account)

Класс Account реализует понятие банковского счёта или кошелька. Основным атрибутом класса являются начальный баланс счёта initBalance типа Cost. Все счета в проекте представлены плоским списком, любой счёт может использоваться как для списания финансовых средств, так и для их пополнения. Счета могут быть ассоциированы с работами, ресурсами, календарями или проектом в целом.

3.6.4 Класс «Пополнение счёта» (Replenishment)

Для учёта доступности финансовых средств каркас предусматривает класс Replenishment, экземпляры которого описывают поступление финансовых средств на счёт. Каждый экземпляр класса Replenishment хранит объектную ссылку на счёт назначения (зачисления), сумму поступлений с указанием валюты (типы Cost и Currency), а также планируемую и актуальную даты поступления финансовых средств на счёт. При этом планируемая дата является обязательным атрибутом, а актуальная — опциональным. Используя приписанные объекты Replenishment можно реконструировать профиль доступных финансовых средств на счете. В этом смысле назначение и организация данного класса аналогична рассмотренному выше классу Supply.

3.6.5 Класс «Исчисление стоимости работы» (TaskRate)

Класс TaskRate позволяет ассоциировать работу и финансовый счет, откуда привлекаются средства для ее выполнения или куда зачисляются средства в случае ее прибыльности. Каждый экземпляр класса хранит объектные ссылки на работу и на финансовый счет, а также значения атрибутов, устанавливающих характер расходов или доходов (атрибут типа CostType), фиксированную или приведенную стоимость выполнения работы (атрибут типа Cost), стоимость прерываний работы (атрибут типа Cost) и профиль финансирования (атрибут типа CostProfile). Опциональными атрибутами класса являются актуальная стоимость и период действия применяемого

тарифа. Данные атрибуты необходимы, чтобы оценить штрафные санкции в случае задержки или опережения работ относительно планируемых дат.

Приведенная стоимость может быть отнесена к единице рабочего или календарного времени, а также к единице трудозатрат работы. Профиль финансирования представляется перечислимым типом CostProfile со значениями AtStart, AtEnd и Uniform, устанавливающими, что средства списываются со счета или зачисляются на счет в начале соответствующего временного периода, в его конце или расходуются равномерно на протяжении всего периода. Примечательно, что с одной и той же работой может быть ассоциировано несколько экземпляров данного класса. Кроме того, следует принимать во внимание, что в роли ассоциированной работы могут быть не только простые активности, но и любые другие виды работ (вехи, гамаки, структуры работ или мультимодальные работы).

3.6.6 Класс «Исчисление стоимости ресурса» (ResourceRate)

Организация класса ResourceRate «Исчисление стоимости ресурса» аналогична рассмотренному выше классу TaskRate за исключением того, что он определяет не стоимость выполнения работы, а стоимость привлечения использования ресурсов при выполнении работ. В данном классе определяются объектные ссылки на соответствующие экземпляры ресурса и счёта, опциональные даты начала и конца действия тарифа, планируемые и актуальные значения стоимости, характер стоимости и её выплаты. С одним ресурсом может быть ассоциировано несколько экземпляров данного класса. Итоговая стоимость использования ресурса в той или иной работе будет определяться как сумма затрат по каждому из тарифов с учётом временных показателей работы. Коллекция объектов ResourceRate с установленными датами действия тарифов позволяет реконструировать всю историю изменений стоимости ресурса в ходе проектной деятельности.

4. Организация классов математических объектов и решателей

Пакеты классов математических объектов (Mathematics) и решателей (Solvers) в определённой степени изолированы от классов прикладных данных (Applications), рассмотренных выше. Данные классы реализуют математические понятия и алгоритмы теории расписаний. Для сведения прикладных задач к обобщенной постановке проектного планирования, формулируемой в математически нейтральной форме, предусмотрены специальные классы редукции (Reductions), которые реализуют интерфейсы математических объектов с учетом особенностей прикладных задач и, тем самым, выполняют функции посредников между прикладными и математическими классами каркаса.

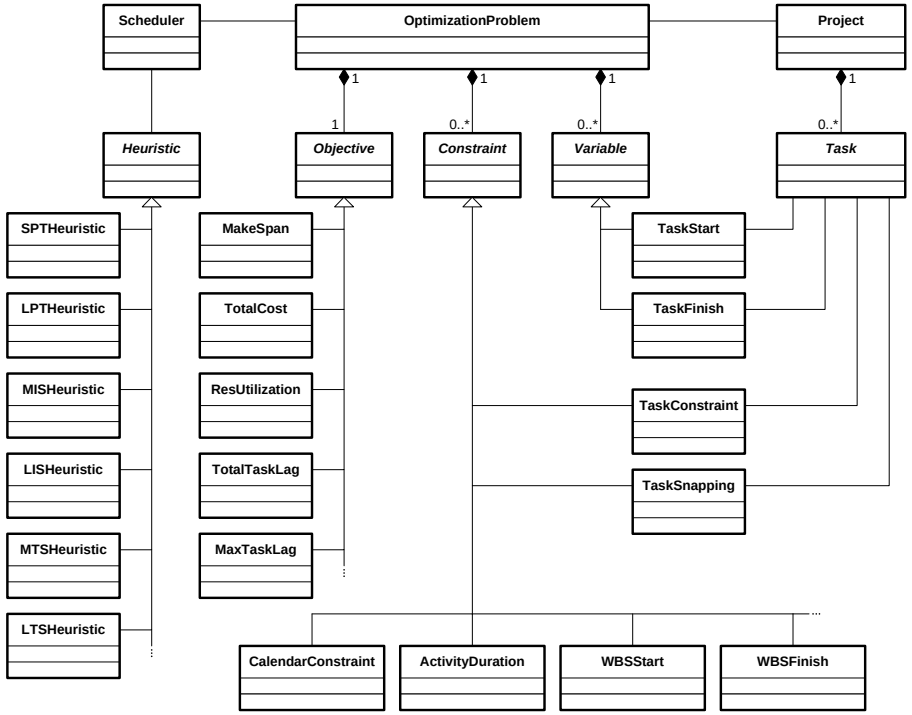

Рис. 8. UML-диаграмма классов математических объектов и решателей
Fig. 8. UML diagram of classes of mathematical objects and solvers

4.1 Класс «Оптимизационная задача» (OptimizationProblem)

Класс OptimizationProblem предназначен для постановки задачи условной нелинейной оптимизации путем задания множества переменных, целевой функции и системы нелинейных алгебраических ограничений. Класс реализуется как композиция всех математических объектов, участвующих в постановке задачи. Такими объектами являются переменные задачи класса Variable, целевая функция класса Objective и алгебраические ограничения класса Constraint.

Поскольку постановка оптимизационной задачи полностью определяется особенностями прикладной задачи проектного планирования, конструирование объектов OptimizationProblem и их наполнение условиями задачи осуществляется в классе Project с помощью вызова метода initOptimizationProblem(). Реализация данного метода предполагает обход всех экземпляров прикладных классов, ассоциируемых с проектом, и вызов соответствующих одноимённых методов для них. В результате каждый экземпляр прикладных классов, в том числе и сам проект, конструирует и инициализирует соответствующие математические объекты и включает их в

композицию класса OptimizationProblem. Заметим, что конструируемые математические объекты являются экземплярами редукционных классов, которые наследуют интерфейсы математических классов Variable, Objective, Constraint и реализуют их с учетом особенностей прикладной задачи. С этой целью они хранят ссылки на соответствующие прикладные данные. Например, для инициализации целевой функции оптимизационной задачи типа Objective создается соответствующий объект, который хранит ссылку на прикладной объект Project и через нее получает доступ ко всем проектным данным, участвующим в вычислении значений целевой функции. Аналогично задаются алгебраические ограничения. Опишем реализацию используемых вспомогательных классов более подробно.

4.2 Класс «Область допустимых значений» (ValueDomain)

Вспомогательный класс ValueDomain предназначен для задания области допустимых значений переменных в решаемой задаче. Поскольку переменные задачи могут выражать разные понятия и описываться разными типами, то наиболее рациональной представляется реализация класса ValueDomain в виде шаблона, параметризуемого типом переменной. В обсуждаемой обобщенной постановке проектного планирования все переменные задачи связаны с временными характеристиками, поэтому применяется конкретная реализация.

Область значений может быть представима отдельными точками, интервалами, бесконечными полуинтервалами, причем все элементы множества не пересекаются и упорядочены. Это обеспечивает эффективность операций проверки принадлежности заданного значения или интервала заданной области, пересечения и объединения заданных областей.

В интерфейсе данного класса для этих целей предусмотрены соответствующие методы:

- boolean isEmpty() — возвращает True, если множество допустимых значений пусто, и False в противном случае;
- boolean contains(ValueDomain&) — возвращает True, если множество допустимых значений полностью содержит заданную область, и False в противном случае;
- ValueDomain unite(ValueDomain&) — объединяет текущую область допустимых значений с другой, переданной в качестве параметра, и возвращает результат как новый экземпляр класса;
- ValueDomain intersect(ValueDomain&) — пересекает текущую область допустимых значений с другой, переданной в качестве параметра, и возвращает результат как новый экземпляр класса.

4.3 Интерфейс «Переменная» (Variable)

Интерфейс Variable определяет методы доступа к переменным задачи проектного планирования, получения, хранения и установки их значений. Используя интерфейс Variable и стандартные шаблоны коллекций, можно определить вектор переменных задачи, например, как ArrayOf<Variable*>.

Поскольку данные переменные связаны с прикладными данными, а именно с временными характеристиками проектных работ, то в составе каркаса предусмотрены конкретные классы TaskStart и TaskFinish, которые наследуя интерфейс Variable, предоставляют доступ к дате и времени начала и завершения работ типа DateAndTime. Кроме заданных значений даты и времени, каждая переменная может принимать неустановленное значение Unset и неизвестное значение Unknown. Например, неизвестное значение присваивается всем переменным перед решением задачи для указания необходимости выполнения соответствующих вычислений. Статус неустановленной переменной означает, что данная переменная была исключена из процесса решения в силу логических условий, связанных с выполнимостью соответствующих работ. Кроме методов доступа к переменным интерфейс Variable определяет метод получения ограничений задачи, в которых данная переменная участвует. Данный метод используется в ходе составления расписаний при анализе и согласованном разрешении наложенных ограничений.

4.4 Интерфейс «Целевая функция» (Objective)

Обсуждаемая обобщенная постановка задач проектного планирования допускает использование альтернативных целевых функций. Поэтому целевая функция в составе каркаса определяется как абстрактный класс с единым интерфейсом, от которого наследуются возможные конкретные реализации, а именно: MakeSpan (время выполнения всего проекта), TotalCost (суммарная стоимость всего проекта), ResUtilization (использование ресурсов), TotalTaskLag (суммарная задержка по всем работам относительно их директивных сроков), MaxTaskLag (максимальная задержка из всех работ относительно директивных сроков) и другие.

Единый интерфейс Objective определяет два основных метода:

- float getValue(ArrayOf<Variable*>&) — возвращает вещественное значение целевой функции по заданному вектору переменных задачи. Реализация метода в конкретных классах должна допускать корректную работу в тех случаях, когда не все переменные имеют предустановленные значения, но может быть вычислена, например, нижняя оценка целевой функции;
- DateAndTime getArgMin(ArrayOf<Variable*>&, int index, ValueDomain&) — возвращает значение заданной переменной, при которой достигается минимум целевой функции на заданной области при фиксированных значениях других переменных.

4.5 Интерфейс «Ограничение» (Constraint)

Принципы организации интерфейса Constraint и наследуемых от него конкретных классов прикладных ограничений во многом аналогичны рассмотренным выше. Интерфейс Constraint определяет следующие методы:

- bool isSatisfied() — возвращает True, если ограничение удовлетворено на ассоциируемом с ним множестве переменных и False в противном случае;
- getVariables(ArrayOf<Variable*>&) — возвращает множество ассоциируемых с ограничением переменных;
- DependencyType getDependency(Variable&) — возвращает одно из значений Independent, Dependent, Vague, определяющее характер зависимости переменной с указанным индексом;
- int getPriority() — возвращает целочисленный приоритет ограничения, используемый при разрешении переопределенных систем;
- ValueDomain resolveFor(ArrayOf<Variable*>&, int index) — возвращает множество значений для заданной переменной, при которых ограничение разрешается при фиксированных других переменных. Предполагается, что реализация данного метода в ограничениях регулярного алгебраического вида должна учитывать переменные с возможным неизвестным состоянием, которые в этом случае интерпретируются как отсутствующие.

Классы прикладных ограничений, наследуя интерфейс Constraint, реализуют данные методы в результате доступа к соответствующим прикладным данным и поэтому отнесены к пакету Reductions. Например, класс CalendarConstraint, ассоциируемый с классом Calendar, реализует ограничение допустимого рабочего времени начала или завершения работы. Классы TaskConstraint и TaskSnapping, ассоциируемые с классом Task, реализуют явные временные условия и правила выравнивания работ, атрибуты которых являются фактическими параметрами порождаемых алгебраических ограничений.

Аналогичным образом реализуются другие классы прикладных ограничений. Класс ActivityDuration определяет строгую алгебраическую зависимость между временами начала и завершения работы с учетом её продолжительности и применяемого календаря. Классы WBSStart и WBSFinish определяют соответствующие зависимости старта и завершения структуры работ от соответствующих параметров дочерних работ. Классы ShortHammockStart, ShortHammockFinish, LongHammockStart и LongHammockFinish определяют аналогичные зависимости «гамаков» от взаимосвязанных предшественников и последователей, классы LinkMinLag и LinkMaxLag — условия предшествования работ с учётом минимального и максимального временного лага, ResourceConstraint — условие доступности ресурса, AccountConstraint — условие наличия средств на счете.

4.6 Интерфейс «Эвристика» (Heuristic)

Для решения задач проектного планирования в обобщенной постановке разработан приближенный алгоритм, основанный на эвристиках [31]. Каркас предоставляет реализацию данного алгоритма в виде соответствующего класса Scheduler. Однако использование последнего предполагает задание эвристик в виде упорядоченного множества объектов соответствующего типа Heuristic.

Интерфейс Heuristic определяет единственный метод:

- VariablePriority compare(Variable& var1, Variable& var2) — возвращает результат сравнения приоритетов пары переменных в виде одного из значений перечислимого типа VariablePriority: FirstOverSecond (первая переменная приоритетнее), SecondOverFirst (вторая переменная приоритетнее), Equal (приоритеты переменных равны). В качестве входных переменных метод принимает ссылки на сравниваемые переменные задачи.

Данный метод реализуется в конкретных классах, наследуемых от интерфейса Heuristic. В частности, каркас предоставляет готовые к использованию реализации следующих эвристик:

- MISHeuristic (Most Immediate Successors) — выбирает переменную с наибольшим количеством непосредственных переменных-последователей;
- LISHeuristic (Least Immediate Successors) — выбирает переменную с наименьшим количеством непосредственных переменных-последователей;
- MTSHeuristic (Most Total Successors) — выбирает переменную с наибольшим количеством всех переменных-последователей;
- LTSHeuristic (Least Total Successors) — выбирает переменную с наименьшим количеством всех переменных-последователей;
- SPTHeuristic (Shortest Process Time) — выбирает переменную с наименьшей разностью значений с переменными-последователями;
- LPTHeuristic (Longest Process Time) — выбирает переменную с наибольшей разностью значений с переменными-последователями.

Более строгое описание эвристик приводится в [31]. Поскольку конкретные классы переменных имеют непосредственную связь с прикладными объектами, формирующими вектор переменных, то становится возможной реализация предметно-ориентированных эвристик с учетом особенностей прикладной задачи и более быстрых способов составления расписаний.

Поскольку применение отдельной эвристики не гарантирует вынесение окончательного вердикта о приоритете одной переменной над другими, на практике применяются иерархические стратегии, состоящие в последовательном применении нескольких эвристик. В каркасе такая

возможность обеспечивается заданием упорядоченного множества разнотипных объектов типа Heuristic.

4.7 Класс «Решатель» (Scheduler)

Scheduler является конкретным классом, реализующий алгоритмы точного и приближенного решения оптимизационной задачи построения расписания. Интерфейс класса определяет пять основных метода:

- boolean state(OptimizationProblem&) — задаёт оптимизационную задачу составления расписания. Возвращает True в случае корректно заданных условий и False в противоположном случае;
- setHeuristics(ArrayOf<Heuristic*>&) — задать упорядоченное множество эвристик для приближенного решения;
- boolean solveApproximately() — ищет решение поставленной задачи, используя описанный приближенный алгоритм с предустановленными эвристиками. Возвращает True, если расписание было успешно составлено и False в противоположном случае;
- boolean solveExactly() — ищет решение заданной оптимизационной задачи, используя точный алгоритм границ и ветвей. Возвращает True в случае успешного поиска решения и False в противоположном случае, например, при исчерпании отведенного процессорного времени;
- generateReport(Report&) — генерирует отчет о процессе работы алгоритмов и качестве найденного приближенного решения.

Рассмотрим листинг программы на языке Си++, иллюстрирующий типовую последовательность вызова методов, а также реализацию основных и вспомогательных методов класса. Основное внимание уделим методу реализации приближенного алгоритма solveApproximately().

```
void main()
    {
    …
    Project project;
    …
    OptimizationProblem problem;
    project. initOptimizationProblem( problem );
    ArrayOf<Heuristic*> heuristics;
    heuristics.push_back( new MISHeuristic() );
    heuristics.push_back( … );
    …
    Scheduler scheduler;
    if ( scheduler.state( problem ) )
        {
```

```
        scheduler. setHeuristics( heuristics );
        if ( scheduler. solveApproximately() )
            {
            ArrayOf<Variable*> &allVariables = m_problem.getVariables();
            for ( int i = 0; i < allVariables.size(); ++i )
                allVariables[ i ].submit();
            }
        }
    Report report;
    scheduler. generateReport( report );
    …
    }

bool Scheduler::state( OptimizationProblem &problem )
    {
    m_problem = &problem;
    return true;
    }

void Scheduler::setHeuristics( ArrayOf<Heuristic*> &heuristics )
    {
    m_heuristics = &heuristics;
    }

bool Scheduler::solveApproximately()
    {
    ArrayOf<Variable*> &allVariables = m_problem.getVariables();
    checkAllVariablesAsUnknown( allVariables );
    ArrayOf<int> activeVariableIndices;
    computeIndependentVariableIndices( activeVariableIndices, allVariables );
    while ( activeVariableIndices.size() > 0 )
        {
        int selectIndex = selectVariable( activeVariableIndices, allVariables );
        if ( ! computeVariable( allVariables, selectIndex ) )
            return false;
        updateActiveVariableIndices( activeVariableIndices, allVariables, selectIndex );
        }
    return true;
    }
```

Вначале метод переводит переменные задачи в неизвестное состояние Unknown вызовом вспомогательного метода setAllVariablesAsUnknown(). Далее определяется множество независимых переменных с помощью метода computeIndependentVariableIndices() и инициализируется множество активных переменных activeVariableIndices. Дальнейшая работа алгоритма предполагает циклическую обработку данного множества. На каждом шаге цикла алгоритм выбирает одну из переменных на основе предустановленных эвристик (метод selectVariable()) и пытается найти её допустимое значение, удовлетворяющее всем ассоциированным с ней ограничениям (метод computeVariable()). Если допустимое значение найти не удаётся, то работа алгоритма прерывается с вердиктом о некорректно заданных условиях поставленной задачи. В случае успешного поиска множество активных переменных обновляется путем исключения найденной переменной и добавления новых переменных, зависящих только от уже обработанных (метод updateActiveVariableIndices()). Ниже приведён листинг перечисленных вспомогательных методов.

```
void Scheduler::checkAllVariablesAsUnknown( ArrayOf<Variable*> &variables ) const
    {
    for ( int i = 0; i < variables.size(); ++i )
        variables[ i ]->checkAsUnknown();
    }

void Scheduler::computeIndependentVariableIndices( ArrayOf<int> &activeVariableIndices,
                                                   const ArrayOf<Variable*> &allVariables ) const
    {
    activeVariableIndices.clear();
    for ( int i = 0; i < allVariables.size(); ++i )
        if ( isNonDependentVariable( allVariables[ i ]->getConstraints(), allVariables[ i ] ) )
            activeVariableIndices.push_back( i );
    }

bool Scheduler::isNonDependentVariable( const ArrayOf<Constraint*> constraints,
                                        const Variable* variable ) const
    {
    for ( int i = 0; i < constraints.size(); ++i )
        if ( constraints[ i ]->getDependency( variable ) == DEPENDENT )
            return false;
    return true;
    }

int Scheduler::selectVariable( const ArrayOf<int> &activeVariableIndices,
                               const ArrayOf<Variable*> &allVariables ) const
```

```
    {
    ArrayOf<int> candidates = activeVariableIndices;
    ArrayOf<int> result;
    for ( int i = 0; i < ( *m_heuristics ).size(); ++i )
        {
        getPriorityVariableIndices( result, candidates, allVariables, ( *m_heuristics )[ i ] );
        if ( result.size() == 1 )
            return ( result[ 0 ] );
        candidatIndices = result;
        }
    return ( result[ 0 ] );
    }

void Scheduler::getPriorityVariableIndices( ArrayOf<int> &result,
                                            const ArrayOf<int> &candidates,
                                            const ArrayOf<Variable*> &allVariables,
                                            Heuristic* heuristic ) const
    {
    result.clear();
    result.push_back( candidates [ 0 ] );
    for ( int i = 1; i < candidates.size(); ++i )
        {
        VariablePriority priority = heuristic->compare( ( *( allVariables[ result[ 0 ] ] ) ),
                                                        ( *( allVariables[ candidates[ i ] ] ) ) );
        if ( priority == FIRST_OVER_SECOND )
            continue;
        if ( priority == SECOND_ OVER_FIRST )
            result.clear();
        result.push_back( candidates[ i ] );
        }
    }

bool Scheduler::computeVariable( ArrayOf<Variable*> &allVariables,
                                 const int &selectedIndex ) const
    {
    ArrayOf<Constraint*> drivingConstraints;
    getDrivingConstraints( drivingConstraints,
                           allVariables[ selectedIndex ]->getConstraints(),
                           allVariables[ selectedIndex ] );
    sortConstraintsByPriority( drivingConstraints );
    if ( drivingConstraints[ 0 ]->canSkipVariable( allVariables[ selectedIndex ] ) )
```

```
        {
        allVariables[ selectedIndex ]->unset();
        return true;
        }
    ValueDomain resultD;
    ValueDomain tempD;
    for ( int i = 0; i < drivingConstraints.size(); ++i )
        {
        tempD = drivingConstraints[ i ]->resolveFor( allVariables, selectedIndex );
        if ( i == 0 )
            {
            if ( tempD.isEmpty() )
                {
                reportError( drivingConstraints[ i ], allVariables[ selectedIndex ] );
                return false;
                }
            else
                resultD = tempD;
            }
        else
            {
            tempD = tempD.intersect( resultD );
            if ( tempD.isEmpty() )
                reportUnresolvedConstraint( drivingConstraints[ i ],
                                            allVariables[ selectedIndex ] );
            else
                resultD = tempD;
            }
        }
    allVariables[ selectedIndex ] = ( m_problem->getObjective() ).getArgMin( allVariables,
                                                                              selectedIndex,
                                                                              resultD );
    return true;
    }

void Scheduler::getDrivingConstraints( ArrayOf<Constraint*> &drivingConstraints,
                                       const ArrayOf<Constraint*> constraints,
                                       const Variable* variable ) const
    {
    drivingConstraints.clear();
    for ( int i = 0; i < constraints.size(); ++i )
```

```
        if ( constraints[ i ]->getDependency( *variable ) != INDEPENDENT )
            drivingConstraints.push_back( constraints[ i ] );
    }

void Scheduler::updateActiveVariableIndices( ArrayOf<int> &activeVariableIndices,
                                             const ArrayOf<Variable*> &allVariables,
                                             const int &selectedIndex ) const
    {
    activeVariableIndices.erase( activeVariableIndices.find( selectedIndex ) );
    const ArrayOf<Constraint*> &constraints = allVariables[ selectedIndex ]->
                                                                    getConstraints();
    for ( int i = 0; i < constraints.size(); ++i )
        if ( constraints[ i ]->getDependency( *( allVariables[ selectedIndex ] ) ) ==
                                                                    INDEPENDENT )
            {
            const ArrayOf<Variable*> &assVars = constraints[ i ]->getVariables();
            for ( int j = 0; j < assVars.size(); ++j )
                if ( constraints[ i ]->getDependency( *( assVars[ j ] ) ) == DEPENDENT )
                    if ( isActiveVariable( assVars[ j ] ) )
                        activeVariableIndices.push_back( allVariables.find( assVars[ j ] ) );
            }
    }

bool Scheduler::isActiveVariable( const Variable* variable ) const
    {
    const ArrayOf<Constraint*> &constraints = variable->getConstraints();
    for ( int i = 0; i < constraints.size(); ++i )
        if ( constraints[ i ]->getDependency( *variable ) == DEPENDENT )
            {
            const ArrayOf<Variable*> &assVars = constraints[ i ]->getVariables();
            for ( int j = 0; j < assVars.size(); ++j )
                if ( constraints[ i ]->getDependency( assVars[ j ] ) == INDEPENDENT )
                    if ( assVars[ j ]->isUnknown() )
                        return false;
            }
    return true;
    }
```

5. Методология разработки приложений теории расписаний на основе SAF-каркаса

Одним из принципиальных требований, предъявляемых к SAF-каркасу, была возможность повторного использования имеющихся модулей при

программной реализации новых моделей, методов и приложений теории расписаний при относительно низких затратах на доработку.

Во многом данное требование удается удовлетворить благодаря принятым в качестве методологической основы принципам объектно-ориентированного программирования и оригинальной многослойной архитектуре каркаса. Рассмотрим их более подробно на примере разработки программных приложений теории расписаний и, в частности, приложений календарно-сетевого планирования.

Каркасом предусматривается довольно развитый набор готовых к использованию модулей и поэтому разработка типового приложения календарно-сетевого планирования, главным образом, сводится к реализации графического интерфейса пользователя (GUI), подключению и конфигурированию имеющихся модулей. В приведенной многослойной архитектуре каркаса элементы GUI составляют самый внешний слой, имеют непосредственный доступ к прикладным данным и могут использовать средства стандартных графических библиотек. Например, популярные GUI библиотеки, такие как Qt, MFC, BCG, предоставляют средства для визуализации проектного плана в виде диаграммы Ганта, построения графиков и диаграмм общего вида, отображения календарей и других данных, характерных для календарно-сетевого планирования.

Для поддержки приложением требуемых функций управления прикладными данными достаточно воспользоваться пакетом Applications, а для решения соответствующих задач составления расписаний — пакетами Reductions, Mathematics и Solvers. Поскольку они предоставляют все необходимые классы для задания условий задач проектного планирования в расширенных постановках и их решения, то реализация данных функций сводится к использованию классов прикладных данных и решателей. Если известны математические особенности прикладной задачи, а к ее решению предъявляются повышенные требования эффективности, то можно сконфигурировать классы решателей соответствующими целевыми функциями и эвристиками, реализации которых также включены в состав каркаса. При этом следует учесть, что выбор эвристик и порядок их применения во многом диктуется целевой функцией оптимизационной задачи.

Таким образом, разработка типового приложения календарно-сетевого планирования с функциями управления данными и решения задач проектного планирования RCPSP в расширенных постановках требует относительно низких затрат, обусловленных, главным образом, следующими работами:

- разработка GUI целевого приложения на основе графических библиотек общего назначения;
- использование имеющихся классов прикладных данных для задания условий задачи планирования;
- использование и конфигурирование имеющих классов решателей соответствующими целевыми функциями и эвристиками для

эффективного приближенного решения поставленной задачи планирования.

В других, более специальных случаях целевых приложений могут потребоваться дополнительные усилия, связанные с выполнением некоторых из перечисленных ниже работ:

- развитие пакета классов Applications для редукции задач теории расписаний к постановке RCPSP;
- развитие пакета классов Applications для представления условий задач RCPSP в расширенных постановках;
- развитие пакета классов Reductions для редукции прикладной задачи к математической постановке условной оптимизации GCPSP;
- развитие пакета классов Solvers для реализации новых точных и приближенных алгоритмов, а также новых эвристик для них.

Обсудим перечисленные возможности разработки целевых приложений теории расписаний в результате развития и конфигурирования классов каркаса.

5.1 Развитие пакета Applications для редукции задач теории расписаний к постановке RCPSP

Обычно задачи теории расписаний не формулируются как задачи проектного планирования, хотя в большинстве случаев могут быть проинтерпретированы в их терминах или сведены к ним за полиномиальное время. В подобных случаях требуется разработка специальных классов прикладных данных, выражающих соответствующие понятия рассматриваемой предметной области и реализующих их, например, с помощью классов пакета Applications.

В качестве примера рассмотрим задачу составления школьного расписания. Будем считать, что в школе занятия проводятся в соответствии с единым регулярным расписанием уроков и перемен, повторяющимся каждый день с понедельника по пятницу. Исключение составляют дни каникул и праздничные дни. В терминах проектного планирования временной аспект функционирования школы описывается с помощью понятия рабочего календаря. Поскольку каркас предоставляет обобщенную реализацию рабочего календаря с регулярными и исключительными правилами, для представления школьного календаря можно воспользоваться соответствующим классом Calendar. Школьный календарь может быть реализован в целевом приложении с помощью конструирования соответствующего объекта и инициализации его атрибутов. Альтернативная реализация состоит в определении класса-наследника и в уточнении рабочих интервалов, регулярных правил и исключений непосредственно в его конструкторе.

У каждого класса учеников есть свой предопределённый образовательными стандартами план занятий на год и на каждый семестр. В терминах проектного

планирования циклы занятий представляются вложенными структурами работ, а каждое занятие или урок в отдельности — активностью с продолжительностью 45 минут. При реализации целевого приложения можно воспользоваться классами WBS и Activity, входящими в состав пакета каркаса Applications, а в целевом приложении построить годовой план школьных занятий. Для упрощенного задания циклов занятий по отдельным предметам и количествам учебных часов можно определить специальные классы-наследники Cycle и Lesson и реализовать в них необходимые вспомогательные методы.

Для проведения занятия в определенное время необходимо, чтобы были свободны ученики класса, а также были доступны учитель и учебный класс. В терминах задачи RCPSP ученики класса, учитель и учебный класс следует рассматривать как возобновимые ресурсы соответствующих типов с доступным единичным количеством. При этом учитель обычно проводит весь цикл занятий для одного класса по одному из своих предметов. А занятия по некоторым предметам требуют специально оборудованных классов. Данные условия в рамках задачи RCPSP задаются путем назначения индивидуальных ресурсов на соответствующие работы. Обычно для организации циклов занятий по отдельным предметам необходимо предусмотреть временные интервалы между ними, например, для подготовки домашних занятий или отдыха учащихся. Данные условия естественным образом интерпретируются в терминах отношений предшествования между работами с соответствующими задержками. Для этих целей может быть определен класс Rule, который наследуя класс каркаса Link, уточняет способ параметризации данных условий.

Таким образом, рассмотренный пример задачи составления школьного расписания может быть сведён к задаче проектного планирования, а разработка программного приложения — к непосредственному использованию классов каркаса или к их наследованию с доопределением вспомогательных методов инициализации прикладных данных. В первом случае может оказаться полезным использование алиасов или предкомпиляторных директив для переименования классов каркаса. Тогда разработка GUI целевого приложения может осуществляться в терминах предметной области, а не проектного планирования. Вместо классов Project, WBS, Activity, Resource, Link можно использовать более естественные для целевого приложения названия School, Cycle, Lesson, Teacher, Room, Class, Rule. Во втором случае реализуются новые предметно-ориентированные классы путем наследования от классов каркаса или в результате их использования с определением нового интерфейса, характерного для условий решаемой прикладной задачи.

5.2 Развитие пакета Applications для представления условий задач RCPSP в расширенных постановках

Хотя пакет каркаса Applications предоставляет довольно развитый набор классов прикладных данных для задания условий RCPSP задач в расширенных постановках, вполне допустимы ситуации, когда требуется еще расширить данный набор, например, для поддержки новых моделей исполнения работ, привлечения ресурсов, специфических типов ограничений и т.п. С математической точки зрения класс задач остается неизменным, но появляется дополнительная специфика, связанная с новыми типами целевых функций и наложенных ограничений. В нашей работе [31] обосновывается возможность математической постановки и решения, так называемых, задач GCPSP с целевыми функциями и наложенными алгебраическими ограничениями общего характера.

В соответствии с объектно-ориентированным подходом развитие пакета Applications может осуществляться различными способами. Например, новые классы прикладных данных можно наследовать от существующих классов с определением новых свойств объектов и уточнением способов параметризации условий прикладных задач. Можно создавать новые классы прикладных данных, определяя их свойства и устанавливая отношения ассоциации, композиции, агрегации с существующими классами каркаса. В отличие от простого переименования классов, упоминаемого в предыдущем разделе, становится возможной реализация обобщенных моделей прикладных данных для задания специальных условий RCPSP задач в расширенных постановках.

В качестве примера, иллюстрирующего необходимость развития пакета Applications, приведем приложение визуального пространственно-временного моделирования проектов. В отличие от традиционных информационных систем календарно-сетевого планирования и управления проектами, составление расписаний в подобных приложениях осуществляется также с учетом пространственных ограничений, регламентирующих условия выполнимости планируемых работ на проектной площадке. Для поддержки пространственных ограничений потребуются дополнительные классы, чтобы представить проектные данные в виде трехмерных геометрических моделей, связать их с проектными работами или порождаемыми ими событиями, а также определить характер пространственных коллизий для составления согласованных расписаний. Примечательно, что реализация классов для задания других видов ограничений, таких как временные условия, отношения предшествования, ресурсные лимиты, рабочие календари, уже предусмотрена типовой конфигурацией пакета Applications и не потребует дополнительных усилий. Поэтому развитие каркаса для обсуждаемых случаев не представляется сложным.

5.3 Развитие пакета Reductions для редукции прикладных задач к постановке GCPSP

Рассмотренные выше способы развития пакета Applications сами по себе не влияют на ход составления расписания алгоритмами, реализации которых уже включены в состав каркаса. Чтобы учесть новые условия задачи проектного планирования, необходимо реализовать соответствующие классы пакета Reductions, с помощью которых данные условия могут быть выражены и проинтерпретированы в математически нейтральных терминах постановки условной оптимизации GCPSP. Программисту, прежде всего, требуется решить, порождают ли введенные или производные прикладные данные новые переменные, меняют ли они вид целевой функции, а также приводят ли они к новым типам алгебраических ограничений. Если это имеет место, то требуется реализовать заново или унаследовать классы, реализующие интерфейсы Variable, ValueDomain, Objective, Constraint, и соответствующим образом модифицировать тело методов формирования условий математической задачи OptimizationProblem. С учетом того, что пакет Reductions содержит реализации всех классов, необходимых для математически корректной редукции задач RCPSP к постановке GCPSP, разработка нескольких дополнительных классов близкой функциональности не потребует значительных усилий.

5.4 Развитие пакета Solvers для реализации новых алгоритмов и эвристик

Пакет Solvers предоставляет готовые к использованию реализации точного и приближенного алгоритмов решения задач GCPSP. В работе [31] формулируются достаточные условия существования решения в данной постановке, а также эквивалентность обобщенного приближенного алгоритма популярному алгоритму последовательной диспетчеризации в случае классической постановки RCPSP. Однако каркас не предоставляет средств анализа существования решений в случае произвольно заданных целевых функций и систем ограничений, а также средств оценки качества найденных приближенных решений. Отчасти вторая проблема нивелируется возможностью поиска точного решения, по крайней мере, для задач низкой размерности. Однако вся ответственность целиком ложится на разработчиков приложений, которые должны обеспечить согласованность задаваемых условий прикладных задач и применяемых алгоритмов.

Одним из способов настройки обобщенного приближенного алгоритма является задание эвристик. Поскольку эвристики реализуются как объекты с общим интерфейсом Heuristic, то разработчики могут предоставить собственные реализации эвристик с учетом прикладных особенностей решаемых задач и сконфигурировать соответствующим образом решатель. В случаях, когда в приложении требуются новые алгоритмы, разработчики

могут их реализовать, основываясь на уже имеющихся в каркасе классах математических объектов и типовых алгоритмов. Примечательно, что используя интерфейсы математических объектов, разработчики, тем самым, предоставляют обобщенные реализации алгоритмов, которые могут быть использованы при решении прикладных задач с произвольными условиями, представимыми математическими объектами наследуемых классов. Однако вопросы конструктивности применения отдельных алгоритмов к конкретным задачам опять же относятся к компетенции разработчиков приложения.

6. Апробация SAF-каркаса

Описанный выше каркас и связанная с ним методология разработки программных приложений теории расписаний были успешно апробированы в ходе построения системы визуального пространственно-временного моделирования проектов Synchro [34]. Данный коммерческий продукт консолидирует в себе традиционные функции систем календарно-сетевого планирования и управления проектами, таких как Oracle Primavera, MS Project, Asta Powerproject, и функции визуального моделирования проектной деятельности.

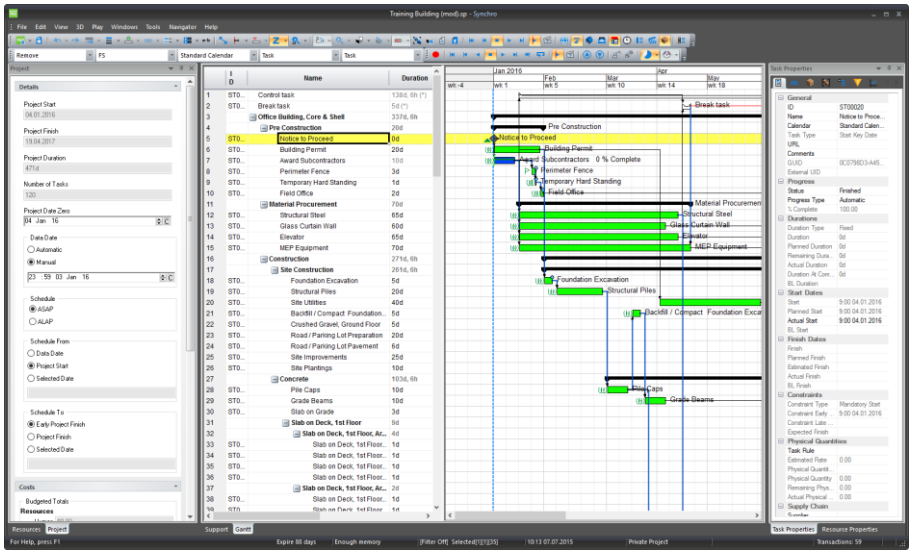

Рис. 9. Графический интерфейс пользователя системы Synchro
Fig. 9. Graphical user interface of the Synchro system

Как следствие, система Synchro должна поддерживать развитый набор пространственных, временных, ресурсных, финансовых ограничений, которые могут быть наложены на проектные работы, а также предоставлять эффективные средства составления согласованных расписаний с учетом всех

видов ограничений. Более того, по мере функциональной эволюции продукта допустим пересмотр данных требований в сторону обобщения и расширения набора ограничений.

На рисунке 9 приведен снимок экрана, иллюстрирующий основные элементы графического интерфейса пользователя системы Synchro, включая диаграмму Ганта, окна задания индивидуальных атрибутов работ, окна задания параметров проектов. Используя последний, пользователь, в частности, может задать некоторые типовые ограничения на даты начала и завершения всего проекта. Временные условия на индивидуальные работы, правила их выравнивания, а также рабочие календари могут быть заданы, используя окно атрибутов работ.

Данные требования удалось удовлетворить в результате использования SAF-каркаса в составе приложения. Каркас не только предоставил готовые к использованию программные средства постановки и решения задач RCPSP, но и обеспечил возможность добавления и поддержки новых видов ограничений в рамках описанной выше методологии.

Проведенные вычислительные эксперименты с большими проектными планами показали высокую эффективность средств каркаса, несмотря на виртуализацию основных вычислительных операций и обобщенную многопараметрическую реализацию моделей прикладных данных. В частности, сравнение с упомянутыми выше популярными приложениями показало, что разработанная система Synchro не уступает им по производительности, однако предоставляет более широкие функциональные возможности, в том числе, из-за поддержки развиваемого набора ограничений.

7. Заключение

Таким образом, обсуждены основные вопросы программной реализации моделей, методов и приложений теории расписаний с использованием SAF-каркаса. Детально рассмотрены принципы организации и функционирования разработанного каркаса, а также его возможности для разработки приложений теории расписаний и, в частности, перспективных систем календарно-сетевого планирования и управления проектами. Успешная апробация каркаса в ходе разработки коммерческого продукта показала правильность принятых проектных решений, а также перспективность использования каркаса для эволюционной разработки серий приложений теории расписаний на единой методологической, программной и инструментальной основе.

Список литературы

[1]. Лазарев А. А., Гафаров Е. Р. Теория расписаний. Задачи и алгоритмы. МГУ им. М. В. Ломоносова, Москва, 2011 г., 222 с.

[2]. Kolisch R., Sprecher A. PSPLIB — A project scheduling library. European Journal of Operational Research, том 96, выпуск 1, 1997 г., с. 205-216.

[3]. Kolisch R., Schwindt C., Sprecher A. Benchmark instances for project scheduling problems. Глава из книги «Handbook on recent advances in project scheduling», под ред. Weglarz J., 1999 г., стр. 197-212.
[4]. Kolisch R., Hartmann S. Heuristic algorithms for solving the resource-constrained project scheduling problem — Classification and computational analysis. Глава из книги «Handbook on recent advances in project scheduling», под ред. Weglarz J., 1999 г., стр. 147-178.
[5]. Hartmann S., Kolisch R. Experimental evaluation of state-of-the-art heuristics for resource constrained project scheduling. European Journal for Operational Research, том 127, выпуск 2, 2000 г., стр. 394-407.
[6]. Kolisch R., Hartmann S. Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update European Journal of Operational Research, том 174, 2006 г., стр. 23-37.
[7]. Lemmen R. Modeling Resource Alternatives in Project Scheduling. Munich University of Applied Sciences. 29 марта 2005 г.
[8]. Интернет-ресурс: «43 полезных сервиса для управления проектами. Без эпитетов». Статья-обзор от 9 февраля 2016, https:habrahabr.ru/post/276873/, дата обращения 25.06.2017
[9]. Щербина О.А. Удовлетворение ограничений и программирование в ограничениях. Интеллектуальные системы. Теория и приложения, том 15, выпуск 1-4, 2011 г., стр. 53-170.
[10]. Creemers T. (et al.) Constraint-based Maintenance Scheduling on an Electric Power Distribution Network Proc. of the 3rd International Conference and Exhibition on Practical Applications of Prolog. Париж, Alinmead Software Ltd., апрель 1995 г.. стр. 135-144.
[11]. Интернет-ресурс: «PLanning Activities on NETworkS» Интернет-сайт разработчика, http://www.iri.upc.edu/research/webprojects/planets/, дата обращения 25.06.2017
[12]. Simonis H., Cornelissens T. Modelling producer/consumer constraints Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995 г., стр. 449-462.
[13]. Интернет-ресурс: «Atlas Venture» Интерней-сайт разработчика, https://atlasventure.com, дата обращения 25.06.2017
[14]. Aggoun A., Gloner Y., Simonis H. Global constraints for scheduling in CHIP. Invited Industrial Presentation. JFPLC 99, 1999 г.
[15]. Glaisner F., Richard L.-M. FORWARD-C: A refinery scheduling system Proc. conf. on Practical Applications of Constraint Technology (PACT97). 1997 г.
[16]. Fromherz M., Gupta V., Saraswat V. Model-based computing: constructing constraint-based software for electro-mechanical systems. Proc. conf. on Practical Applications of Constraint Technology (PACT95). 1995 г., стр. 63-66.
[17]. Baues G., Kay P., Charlier P. Constraint based resource allocation for airline crew management. Proc. ATTIS'94. 1994 г.
[18]. Collignon C. Gestion optimisee de ressources humaines pour l'audiovisuel. Proc. CHIP users' club. 1996 г.
[19]. Интернет-ресурс: «COSYTEC» Интернет-сайт разработчика, http://www.cosytec.com/constraint_programming/cases_studies/administration.htm, дата обращения 25.06.2017

[20]. Simonis H., Charlier P. Cobra — a system for train crew scheduling. Proc. DIMACS workshop on constraint programming and large scale combinatorial optimization. 1998 г.
[21]. Chew T., David J.-M. A constraint-based spreadsheet for cooperative production planning. Proc. AAAI SIGMAN workshop on knowledge-based production planning, scheduling and control. 1992 г.
[22]. Shvetsov I., Kornienko V., Preis S. Interval spreadsheet for problems of financial planning. Proc. PACT97. 1997 г., стр. 373-385.
[23]. Fruhwirth T., Brisset P. Optimal planning of digital cordless telecommunication systems. Proc. PACT97. 1997 г.
[24]. Shih-Ming Chena, F.H. (Bud) Griffisb, Po-Han Chenc, Luh-Maan Chang. A framework for an automated and integrated project scheduling and management system. Automation in Construction, том 35, 2013 г., стр. 89-110.
[25]. Jan Tulke, Mohamed Nour, Karl Beucke. A Dynamic Framework for Construction Scheduling based on BIM using IFC. IABSE Congress Report, 17th Congress of IABSE. 2008 г., стр. 158-159.
[26]. Интернет-ресурс: «ISO 16739:2013» Интернет-страница описания стандарта, http://www.iso.org/iso/catalogue_detail.htm?csnumber=51622, дата обращения 25.06.2017
[27]. Интернет-ресурс: «Фреймворк» Интернет-страница электронной энциклопедии, https://ru.wikipedia.org/wiki/Фреймворк, дата обращения 25.06.2017
[28]. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, технологии и CASE-средства программирования. Киев, Наукова думка, 2013 г., 283 с.
[29]. Горбунов-Посадов М.М. Расширяемые программы. Москва, Полиптих, 1999 г., 336 с.
[30]. Интернет-ресурс: Martin Fowler: InversionOfControl. Статья-исследование, https://martinfowler.com/bliki/InversionOfControl.html, дата обращения 25.06.2017
[31]. Аничкин А.С., Семенов В.А. Математическая формализация задач проектного планирования в расширенной постановке. Труды ИСП РАН, том 29, выпуск 2, 2017 г., стр. 231-256. DOI: 10.15514/ISPRAS-2017-29(2)-9
[32]. Аничкин А.С., Семенов В.А. Современные модели и методы теории расписаний и календарно-сетевого планирования. Труды ИСП РАН, том 26, выпуск 3, 2014, стр. 212-262. ISSN 2220-6426
[33]. Brucker P., Knust S. Complex scheduling. Springer-Verlag, Berlin, Heidelberg, Germany, 2006 г., 292 с.
[34]. Интернет-ресурс: «Synchro Software» Официальный Интернет-сайт продукта Synchro, http://synchroltd.com, дата обращения 25.06.2017

Object-oriented framework for software development of scheduling applications

[1] A.S. Anichkin <anton.anichkin@ispras.ru>
[1,2] V.A. Semenov <sem@ispras.ru>
[1] Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

Abstract. Theory of scheduling and project planning is widely applied in diverse scientific and industrial areas. To effectively solve application-specific problems, it is necessary to state right objectives as well as to take into account a lot of factors, such as task execution models, precedence relationship between tasks, resource limitations, directive deadlines, working calendars, conditions for financial and logistics support of project tasks, specific spatio-temporal requirements, et al. Therefore, the development of scheduling applications becomes more and more complicated purposes, risky and costly ones. In this paper, we present an innovative object-oriented Scheduling Application Framework (SAF) designed to simplify and accelerate the software development processes. The presented SAF framework is a system of C++ classes that implement basic abstractions of the scheduling theory as well as provide ready-to-use components to build target applications of typical scheduling functionality. As a general-purpose mathematical library, the framework enables to set and solve so-called RCPSP problems (Resource-Constrained Project Scheduling Problem) in extended statements peculiar to popular project management systems. Branch and bound and linear dispatching algorithms have been implemented and included as a part of the framework. A dozen of heuristics has been implemented and has been provided by the framework too to solve large-scale problems more effectively. Thereby target application developers can adjust the application solver properly taking into account application-specific issues and making the search of suboptimum schedules more effective. As a software toolkit the framework enables developers to implement own components and to configure target applications in unified and flexible manner. Due to object-oriented paradigm, multi-layer architecture and class package organization, the application development takes relatively small efforts. The SAF framework has been successively validated during development of a software application intended for visual modeling and planning of projects under diverse spatial-temporal, resource and finance constraints. Due to achieved advantages, the framework looks promising for development of both sophisticated multi-disciplinary systems and effective domain-specific scheduling applications.

Keywords: scheduling theory; project planning and scheduling; software application framework.

DOI: 10.15514/ISPRAS-2017-29(3)-14

For citation: Anichkin A.S., Semenov V.A. Object-oriented framework for software development of scheduling applications. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017. pp. 247-296 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-14

References

[1]. Lazarev A. A., Gafarov E. R. Scheduling theory. Tasks and algorithms. Lomonosov Moscow State University, Moscow, 2011, 222 p (in Russian).

[2]. Kolisch R., Sprecher A. PSPLIB — A project scheduling library. European Journal of Operational Research, vol. 96, issue 1, 1997, pp. 205-216.

[3]. Kolisch R., Schwindt C., Sprecher A. Benchmark instances for project scheduling problems. Chapter in the book «Handbook on recent advances in project scheduling», ed. Weglarz J., 1999, pp. 197-212.

[4]. Kolisch R., Hartmann S. Heuristic algorithms for solving the resource-constrained project scheduling problem — Classification and computational analysis. Chapter in the book «Handbook on recent advances in project scheduling», ed. Weglarz J., 1999, pp. 147-178.
[5]. Hartmann S., Kolisch R. Experimental evaluation of state-of-the-art heuristics for resource constrained project scheduling. European Journal for Operational Research, vol. 127, issue 2, 2000, pp. 394-407.
[6]. Kolisch R., Hartmann S. Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update European Journal of Operational Research, vol. 174, 2006, pp. 23-37.
[7]. Lemmen R. Modeling Resource Alternatives in Project Scheduling. Munich University of Applied Sciences. March 29, 2005.
[8]. Internet: [«43 useful services for project management. Without epithets».] Article-review, February 9, 2016, https://habrahabr.ru/post/276873/, accessed 25.06.2017
[9]. Shcherbina O.A. Satisfaction of constraints and programming in constraints. Intellektualnyie sistemyi. Teoriya i prilozheniya. [Intellectual systems. Theory and applications] Vol. 15, issue 1-4, 2011, pp. 53-170 (in Russian).
[10]. Creemers T. (et al.) Constraint-based Maintenance Scheduling on an Electric Power Distribution Network Proc. of the 3rd International Conference and Exhibition on Practical Applications of Prolog. Paris, Alinmead Software Ltd., April 1995, pp. 135-144.
[11]. Internet: «PLanning Activities on NETworkS» Website of developer, http://www.iri.upc.edu/research/webprojects/planets/, accessed 25.06.2017
[12]. Simonis H., Cornelissens T. Modelling producer/consumer constraints Proceedings 1st Int. Conference on Principles and Practice of Constraint Programming (CP95). Springer-Verlag, LNCS 976, 1995, pp. 449-462.
[13]. Internet: «Atlas Venture» Website of developer, https://atlasventure.com, accessed 25.06.2017
[14]. Aggoun A., Gloner Y., Simonis H. Global constraints for scheduling in CHIP. Invited Industrial Presentation, JFPLC 99. 1999.
[15]. Glaisner F., Richard L.-M. FORWARD-C: A refinery scheduling system Proc. conf. on Practical Applications of Constraint Technology (PACT97). 1997.
[16]. Fromherz M., Gupta V., Saraswat V. Model-based computing: constructing constraint-based software for electro-mechanical systems. Proc. conf. on Practical Applications of Constraint Technology (PACT95). 1995, pp. 63-66.
[17]. Baues G., Kay P., Charlier P. Constraint based resource allocation for airline crew management. Proc. ATTIS'94. 1994.
[18]. Collignon C. Gestion optimisee de ressources humaines pour l'audiovisuel. Proc. CHIP users' club. 1996 г.
[19]. Internet: «COSYTEC» Website of developer, http://www.cosytec.com/constraint_programming/cases_studies/administration.htm, accessed 25.06.2017
[20]. Simonis H., Charlier P. Cobra — a system for train crew scheduling. Proc. DIMACS workshop on constraint programming and large scale combinatorial optimization. 1998.
[21]. Chew T., David J.-M. A constraint-based spreadsheet for cooperative production planning. Proc. AAAI SIGMAN workshop on knowledge-based production planning, scheduling and control. 1992.
[22]. Shvetsov I., Kornienko V., Preis S. Interval spreadsheet for problems of financial planning. Proc. PACT97. 1997, pp. 373-385.

[23]. Fruhwirth T., Brisset P. Optimal planning of digital cordless telecommunication systems. Proc. PACT97. 1997.
[24]. Shih-Ming Chena, F.H. (Bud) Griffisb, Po-Han Chenc, Luh-Maan Chang. A framework for an automated and integrated project scheduling and management system. Automation in Construction, vol. 35, 2013, pp. 89-110.
[25]. Jan Tulke, Mohamed Nour, Karl Beucke. A Dynamic Framework for Construction Scheduling based on BIM using IFC. IABSE Congress Report, 17th Congress of IABSE. 2008, pp. 158-159.
[26]. Internet: «ISO 16739:2013» Web-page of description of the standard, http://www.iso.org/iso/catalogue_detail.htm?csnumber=51622, accessed 25.06.2017
[27]. Internet: «Software framework» Web-page of the electronic encyclopedia, https://en.wikipedia.org/wiki/Software_framework, accessed 25.06.2017
[28]. Lavrischeva E.M Software Engineering of computer systems. Paradigms, technologies and CASE-programming tools. Kiev, Naukova dumka, 2013, 283 p. (in Russian).
[29]. Gorbunov-Possadov M.M. Extensible programs. Moscow, Poliptih, 1999, 336 p. (in Russian).
[30]. Internet: Martin Fowler. InversionOfControl. Research article, https://martinfowler.com/bliki/InversionOfControl.html, accessed 25.06.2017
[31]. Anichkin A.S., Semenov V.A. Mathematical formalization of project scheduling problems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 231-256 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-9
[32]. Anichkin A.S., Semenov V.A. A survey of emerging models and methods of scheduling. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 3, 2014, pp. 5-50 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-1
[33]. Brucker P., Knust S. Complex scheduling. Springer-Verlag, Berlin, Heidelberg, Germany, 2006, 292 p.
[34]. Internet: «Synchro Software» Official website of the product Synchro, http://synchroltd.com, accessed 25.06.2017