Дипломная работа

2 июня 2013 г.

1 Введение

За последних два десятилетия в научном сообществе компьютерное моделирование, названное e-Sience, стало незаменимой частью исследовательского процесса наравне с традиционными инструментами, такими как эмпирические, основанное на экспериментальных наблюдениях, теоретическое моделирование. Современная парадигма (e-Science) связана с развитием инструментария распределенных вычислений для научных исследований, позволяющего консолидировать вычислительные и программные ресурсы для решения сложных мультидисциплинарных задач в форме так называемых композитных приложений.

Термин e-Sience, возникший первоначально в Великобритании, где крупные исследовательские проекты в этой области начались в 2001г. Именно там было дано первое определение e-Science, получившее в дальнейшем широкое распространение: "научно-технологическая область, в которой всевозрастающую роль играет распределенное глобальное взаимодействие посредством сети интернет, с использованием очень больших коллекций данных, компьютерных ресурсов

тера-уровня и высококачественной визуализации, доступных индивидуальному пользователю". В русском языке термин e-Science существует пока преимущественно в англоязычном варианте.
Необходимость в ,что кроме «обычной» информации, размещенной в интернете, специалисты, работающие в сфере науки и образования, нуждаются в доступе к крупномасштабным информационным массивам, базам данных, имеющим объемы памяти, измеряемые терабайтами. Работа с такими массивами требует вычислительных мощностей с производительностью уровня терафлоп. Задача e-Science, таким образом, — создание организационных и технологических структур, разработка соответствующего программного обеспечения для функционирования новой информационной среды с распределенными ресурсами (информационными и вычислительными), обеспечивающими доступ к ним индивидуальных пользователей, исследовательских групп, лабораторий и институтов. Основное русло реализации задач e-Science прокладывают грид-технологии. Эта концепция (нередко ее называют Grid Computing — распределенные сети вычислительных ресурсов) соответствует одному из ведущих и перспективных направлений развития информационно-коммуникационные технологий. Наиболее распространенным подходом к представлению композитных приложений является формализм workflow.

Термин “workflow” используется в двух аспектах — как формальное представление некоторого процесса и как некоторый подход к автоматизации процессов, основанный на подобном представлении.
Начнем с первого аспекта. Буквальный перевод термина “workflow” как «поток работ» плохо раскрывает содержание данного понятия, поэтому зачастую используется термин «сценарий». Под workflow подразумевается формальное представление (модель) некоторого процесса, включающее в себя:

- Описание элементарных операций, из которых состоит процесс.

- Описание исполнителей, которые выполняют указанные операции.

- Описание зависимостей между операциями, а именно — потоков управления, которые определяют последовательность выполнения операций и синхронизацию между ними, и потоков данных, которые определяют передачу информации между операциями.

- Описание внешних событий, которые могут влиять на ход про-

цесса, и правил их обработки.

Второй аспект термина “workflow” нашел отражение в определении, данном в Workflow Management Coalition [2], — « это автоматизация, полностью или частично, бизнес-процесса, при которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил ». Данное определение, несмотря на привязку к бизнес-процессам, хорошо отражает суть workflow-методологии как некоторого подхода к автоматизации вообще говоря различных процессов. Системой управления сценариями (workflow management system, WFMS) будем называть систему, позволяющую создавать сценарии, запускать и управлять их выполнением [2]. WFMS состоит из набора программных компонентов, предназначенных для хранения и интерпретации описаний процессов (сценариев), создания и управления экземплярами запущенных процессов, а также организации их взаимодействия с участниками процесса и внешними приложениями. Программное приложение, непосредственно выполняющее интерпретацию и запуск сценария, а также управляющее экземплярами запущенных процессов, будем называть средой выполнения сценариев (workflow engine).

Приведем основные отличия научных вычислительных процессов от бизнес-процессов:

- Ориентация на сложные вычислительные ресурсы.
- Количество ресурсов, которые потребуются для выполнения сценария (решения задачи), может быть неизвестно априори, так как для некоторых классов задач трудно оценить предстоящий объем вычислений.
- Необходимость работы в динамичной распределенной среде, в которой ресурсы не известны априори и могут быть подвержены отказам.
- Работа с большими объемами данных.
- Необходимость выполнять большое количество идентичных заданий с переменными параметрами.
- Необходимость следить за выполнением процесса и контролировать его, в том числе внося специальные для конкретных случаев изменения.

- Для многих научных workflow характерны иерархии вложенных workflow, создаваемых и уничтожаемых по необходимости.

Указанные особенности научных вычислительных процессов определяют требования, которым должна удовлетворять WFMS, подходящая для описания и выполнения данных процессов в виде сценариев. Традиционные WFMS, рассчитанные на работу с бизнес-процессами, в подавляющем большинстве не подходят для решения научных задач. Поэтому, требуется разработка новых, научных WFMS, с одной стороны опирающихся на сформировавшуюся workflow-методологию, а с другой стороны — специально рассчитанных на требования научных приложений. Пожалуй, главное, что могут почерпнуть научные WFMS из накопленного в данной области опыта, это способы формального представления workflow. В следующем разделе мы рассмотрим основные подходы к представлению сценариев, ссылаясь на уже созданных научных WFMS для того, чтобы одновременно дать обзор существующих решений в области научных вычислительных workflow.

Различные подходы к представлению workflow

За время существования методологии workflow возникло несколько различных подходов их формального описания:

- Использование скриптовых языков.
- Использование графов:
  - ориентированный ациклические графы(DAG);
  - сети Петри.

Рассматриваемые модели будут ориентированы на потоки данных (data-flow), как наиболее применяемые для научных вычислений. При этом основное внимание будет уделено тому, как данный подход позволяет представить workflow в формальном виде или графически представить пользователю.

1.0.1 Скриптовые языки

Скрипт(script) представляет собой набор команд, предназначенный для выполнения интерпретатором без вмешательства пользователя.

Скриптовые языки в качестве средства представления сценариев могут быть удобны пользователям, имеющим опыт программирования. Но не смотря навысокий уровень и простоту, они не достаточно наглядны и интуитивны для обычных пользователей.

1.0.2 Представление сценариев в виде графов

Другой способ представления workflow - это изображение в виде графа. Изначально, графы - это чисто математическая абстракция, но тем не менее, они удобны для неподготовленного пользователя, поскольку представляют workflow наглядно. Правда с увеличением сложности workflow графы тоже усложняются и их становится тяжелее просматривать, тем не менее, наглядность можно сохранить, используя иерархическое представление графа, позволяющее скрывать детали отдельных подграфов.
Для представления workflow наиболее часто используются два типа графов: ориентированные ациклические графы(DAG) и сети Петри.

Ориентированным ациклическим графом называется любой ориентированный граф, в котором нет ориентированных циклов [8]. Вершинами графа являются, например, исполняемые программы или выполняемые операции, а ребра устанавливают зависимости между ними. Преимущество таких графов — простота структуры и реализации. Но есть и недостатки: они накладывают ограничения на типы сценариев — например, нельзя явно задать циклы без применения дополнительных конструкций, уже не связанных с графовым представлением. Кроме того, такие графы способны описывать только модель поведения процесса, не фиксируя его состояние во время выполнения.

Сети Петри — особый класс ориентированных графов. Теория сетей Петри является хорошо известным и популярным формализмом, предназначенным для работы с параллельными и асинхронными системами. Основанная в начале 60-х гг. немецким математиком К. А. Петри, в настоящее время она содержит большое количество моделей, методов и средств анализа.

Рассмотрим основные преимущества сетей Петри при представлении сценариев.
Графическая природа. Сети Петри — графический язык. Поэтому они интуитивно понятны и легки для изучения. Их графическая природа также удобна для взаимодействия с конечными пользовате-

лями.

Формальность описания. Сценарий, описанный в терминах сети Петри, задается строго и точно, потому что семантика классических сетей Петри, как и некоторых дополнений к ним (цвет, время, иерархия) были введены формально.

Выразительность. Сети Петри поддерживают все базисные элементы, необходимые для описания сценария. С их помощью могут быть смоделированы все управляющие конструкции, существующие в современных системах управления сценариями. Более того, точное представление состояний сценария позволяет описывать ситуации неявного выбора и сохранять промежуточные состояния, с возможностью возвращения к ним.

Свойства. В последние десятилетия были подробно изучены основные свойства сетей Петри. Прочные математические основания позволяют делать строгие выводы из этих свойств.

Анализ. Сети Петри отличаются наличием большого числа методов анализа. Это их ценное преимущество с точки зрения использования для описания сценариев — данные методы могут быть использованы для доказательства различных свойств (выполнимости, инвариантности, мертвых переходов и т. д.) и для вычисления характеристик выполнения сценариев (время отклика, время ожидания, степень занятости и пр.). Таким образом, становится возможным оценивать альтернативные сценарии, используя традиционные инструменты анализа сетей Петри.

Подробно сети Петри , их свойства и средства анализа будут описаны в следующей главе.

1.0.3 Модели, ориентированные на потоки данных

Помимо уже упомянутых методов, существует еще один подход, в котором для представления сценариев могут использоваться как скриптовые языки, так и графы, но который несколько отличается от обсуждавшихся ранее подходов своей спецификой. Речь идет о системах управления научными вычислительными сценариями, ориентированными на потоки данных. Как уже было сказано, в научных приложениях часто все необходимые действия сводятся к различным операциям над данными, т. е. в подобных процессах потоки управления и потоки данных совпадают. Поэтому нет необходимости вводить специальные элементы языка для описания логических конструкций,

а достаточно просто обеспечить средства объединения элементарных модулей обработки данных в сеть. Каждый модуль имеет один или несколько входов и выходов, дуги сети соответствуют соединениям выхода одного модуля с входом другого, по которым осуществляется передача данных между модулями. Как только на вход модуля поступили все необходимые данные, происходит запуск программного кода модуля, который производит обработку входных данных, после чего полученные результаты (выходные данные) помещаются в выходы модуля и передаются по соединениям на вход других модулей.

1.0.4 Модели управления запуском workflow

Большинство моделей управления workflow можно разделить на два класса: модели ориентированные на потоки данных(data-flows) и модели ориентированные на потоки управления(control-flows). Оба класса определяют взаимодействие между отдельными задачами-компонентами workflow, но различаются принципами реализации этого взаимодействия.
В workflow, основанных на принципе control-flow, связи между элементами workflow представляет передачу управления от одного задания следующему. Это позволяет формировать внутри workflow такие типы структур, как: последовательное выполнение, параллельное выполнение, циклическое выполнение и условные переходы. в workflow, основанные на принципе data-flow, зависимости между элементами workflow, определяют направление потоков данных от

Так же существуют гибридные системы управления workflow, сочетающие в себе принципы обоих приведённых выше классов. Гибридные системы поддерживают оба типа зависимостей между компонентами workflow, но один из типов, как правило, является доминирующим, а другой используется при необходимости в особых случаях. Например, в data-flows системе такой как Triana, возникают ситуации, когда необходимо последовательно связать задачу, не производящую никакие данные, с задачей, не требующей данных на вход. В таком случае, на этом участке будет использован переход к control-flow зависимости.

Существует много техник построения расписаний запуска workflow. Но почти все они применимы, только если worklow представим в виде направленного ациклического графа (DAG) заданий. В случае,

если workflow содержит циклы, параллельные циклы или условные переходы, применяются методы приведения графа заданий workflow к требуемому виду. С подобной проблемой преобразования графа задач уже сталкивались при разработке компиляторов, поддерживающих автоматическую параллелизацию, и были выработаны техники развёртывания параллельных циклов, устранения обычных циклов, предсказаний при условных переходов.

1.1 Мотивация

В работе будет разработана абстрактная модель workflow, позволяющая В своей работе я разрабатываю модель workflow , объединяющую оба принципа, с целью получения более естественного способа описания реальных инженерных и научных задач. **Задачи исследования:**

- разработка метода унифицированного описания прикладных ПМ1 в распределенной вычислительной среде на основе предметно-ориентированного языка;

- разработка метода унифицированного описания workflow, консолидирующего прикладные программные модули и обеспечивающего организацию взаимодействия между ними, а также метода интерпретации описания в форму, исполнимую в распределенной среде;

  разработка технологии создания и исполнения интерактивных КП на основе унифицированного описания структуры и прикладных ПМ в составе WF; программная реализация разработанных методов, моделей и алгоритмов в рамках платформы облачных вычислений для задач eScience; оценка работоспособности и эффективности разработанного решения на основе вычислительных экспериментов в области моделирования: наноразмерных атомно-молекулярных

  Предметом исследования являются технологии проектирования и разработки композитных приложений в распределенных высокопроизводительных вы- числительных средах. Целью работы является решение важной в создании распределенных систем для научных исследований задачи обоснования, разработки и исследования моделей, методов и алгоритмов для проектирования,

создания и исполнения композитных приложений в распределенной вычислительной среде.

Многие аспекты в данной работе или ограничения приняты с учётом дальнейшей реализации WFMS. Предполагаемая модель которого

Модель создавалась с расчётом на то, что компоненты workflow будут задачи в составе workflow будут

2 Базовые понятия и определения

Определение: Состояние *Состояние любой системы в любой определённый момент времени характеризует, как система будет реагировать на входные данные. Формально, под состоянием мы будем подразумевать всю накопленную системой информацию, которая влияет на поведение системы в текущий момент и в дальнейшие.*

3 Формальное описание модели блок-ориентирова workflow

В данном разделе будут введены базовые понятия, используемые в работе, в том числе понятие атомарного блока,составного блока, графа связей, состояния workflow.

3.1 Программный модулей

Программным модулем будем называть некоторый адаптер над программой или запускаемым скриптом, имеющий унифицированный интерфейс и рассчитанный на работу в распределённой вычислительной среде и поддерживающий средства коммуникации этой распределённой среды.

Унифицированность интерфейса заключается в том:

- Программный модуль начинает
- перед началом запуском, набор входных и выходных портов программного модуля фиксируется.

что модуль запускается один раз при исполнении workflow на заранее выбранном узле(node). распределённой среды.

3.2 Атомарный блок

Атомарный блок является базовой логическим и исполнительным компонентом workflow в нашей модели. Атомарный блок является моделью отдельного программным модуля. Считаем, что модель поведения программного модуля можно формализовать в виде недетерминированным конечного автоматом.

Автоматом называется система , выходы которой зависят не только от поступивших входов, то и от текущего состояния системы. Состояние системы может быть обозначено переменной состояния $s \in \Sigma$, где Σ - это набор всех возможных состояний системы. Конечным автоматом называется автомат, для которого число состояний Σ конечно.

Классическая теория конечных автоматов (Hopcroft and Ullman, 1979) различает два вида автоматов: **Автомат Мили** и **Автомат Мура**. В Автомате Мили выходное значения сигнала явно зависит только от входных значений, в отличии от Автомата Мура, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата. Для полного задания автомата Мили или Мура дополнительно к законам функционирования, необходимо указать начальное состояние и определить внутренний, входной и выходной алфавиты. Между автоматами Мили и Мура существует соответствие, позволяющее преобразовать закон функционирования одного из них в другой или обратно. Автомат Мура можно рассматривать как частный случай автомата Мили, имея в виду, что последовательность состояний выходов автомата Мили опережает на один такт последовательность состояний выходов автомата Мура, т.е различие между автоматами Мили и Мура состоит в том, что в автоматах Мили состояние выхода возникает одновременно с вызывающим его состоянием входа, а в автоматах Мура - с задержкой на один такт, т.к в автоматах Мура входные сигналы изменяют только состояние автомата.

Определим недетерминированный автомата атомарного блока, который будет использоваться в работе.

Определение: Конечный автомат блока *Конечным автоматом блока называется набор* $M = (\Sigma, I, O \cup done, T, s_0)$ *, где*

- Σ *-набор конечных состояния,*

- I *- непустой набор доступных входных портов блока,*

- O *- набор доступных выходных портов блока,*

- $I \bigcap O = \varnothing$ -

- $s_0 \in \Sigma$ *- начальное состояние,*

- $T : \Sigma \times (2^I \backslash \{\varnothing\}) \to 2^{\Sigma \times (2^O \cup \{done\})}$ *отображение сопоставляющее каждому состоянию и набору входных портов набор состояний с соответствующим набором выходных портов.*

В рассматриваемой модели мы абстрагируемся от типизации данных передаваемых между блоками, считая что для любых двух соединённых блоков типы данных для каждой пары соединённых портов совместимы. Поэтому мы сразу можем представлять передаваемые по связям данные как сигналы.

Такой конечный автомат позволяет моделировать сильный недетерминизм в поведении программного модуля.

Если для заданного состояния $s' \in \Sigma$ и непустого набора входных портов $in \in 2^I \backslash \{\varnothing\}$ отображение T(s', in) не пусто, то результат отображения T(s', in) является набором возможных вариантов работы моделируемого программного модуля, с поглощением сигналов(данных) на входных портах и соответствующим переходом в новое состояние и испусканием сигналов по указанным выходным портам. Если же T(s', in) даёт пусто множество, считается, что условия запуска блока из состояния s' не соблюдены и начало работы программного модуля невозможно.

Так как используемая модель имеет тип data-flow, то наборы входных и выходных портов блока должны быть не пусты, чтобы можно было организовать взаимодействие между ними. Таким образом, даже если задача, исполняемая программным модулем,не требует никаких входных данных, необходим какой-либо входной порт, получив сигнал по которому можно было бы начать работу. Для удобства и общности в таких случаях будем вводить ему под именем

do. Так же считаем, что у блока всегда есть выходной порт *done*, который испускает сигнал при каждом переходе.

Как уже было определено выше, у блока имеется **начальное состояние** s_0, т.е. то в котором автомат блока находится перед запуском.

Визуально состояния соединены переходами, рядом с которыми указано, какие порты необходимы для срабатывания перехода и сигналы по каким портам будут испущены.

3.3 Представление workflow

Граф связей workflow *Мультиграф связей workflow* $WFG = (A, D)$ *состоит из набора блоков* A *и набора связей* D*, соединяющих блоки через порты. При этом в состав набора блоков всегда* A *входят два уникальных абстрактных блока Source и Stock, вводимых для задания точек запуска и завершения workflow. Формально связь* $d \in D$ *представима в виде четвёрки* (s, sp, t, tp)*, где* $s, tinA$ *- идентификаторы входной и выходных блоков,* $sp \in out(s), tp \in in(p)$ *- выходной и входной порты соответствующих блоков.*

Для каждого блока $a \in A$ наборы входных портов $p_{in} = in(a)$ и выходных $p_{out} = out(a)$ считаем фиксированными и они соответствуют параметрам модели конечного автомата блока.

Опишем некоторые свойства workflow, связанные со связями и передачей сигналов по ним:

- Связь может быть установлена только между выходным портом одного блока и входным портом другого блока или самого себя.

- Связи могут быть построены как из одного выходного порта ко многим входным, так и из нескольких выходных в один входной порт.

- Если блок испускает сигнал по какому либо порту, то сигнал распространяется по всем связям исходящих из этого порта до конечного порта назначения, таким образом порождаются конкурирующие потоки.

- Входные и выходные порты блоков могут оставаться неподключенными. В случае выходных портов считаем, что сигнал теряется.

4 Возможные ошибки при построении workflow

Рассмотрим возможные ошибки при построении workflow, которые можно выявить из структуры workflow на этапе проектирования.

Так как мы подразумеваем возможность асинхронной работы программных модулей на отдельных участках workflow, то как и в многопоточных приложениях возможно возникновение *состояние гонки*(Race condition). В программировании состояние гонки определяется как ситуация, когда несколько потоков одновременно обращаются к одному и тому же ресурсу, причём хотя бы один из потоков выполняет операцию записи, и порядок этих обращений точно не определен. Ошибки, вызванные состояниями гонки, обычно трудно обнаружить и исправить.

В контексте нашей модели workflow состояние гонки бывает двух типов:

- Когда на один входной порт могут прийти два сигнала одновременно.
- Когда на входные порты блока одновременно приходит такой набор сигналов, что запуск блока возможен по разным наборам сигналов.

4.1 Принципы моделирования работы workflow

Алгоритм моделирования работы workflow будет использоваться для нахождения структурных ошибок в workflow, и для построения графа состояний workflow, который будет использоваться при дальнейшем анализе.

Так как мы хотим моделировать не зная априори время работы каждого блока, а только может быть некоторую оценочную её величину, то можем Моделировать работу workflow будем в дискретном режиме.

Понятие разбиения вводится для формализации конкуренции между сигналами, распространяющимися по связям внутри workflow.

Опредление: Разбиение *Разбиением назовём пару (fork, splitting), где fork - имя блока, которой испустившего набор сигналов, а splitting - кортеж состоящий из n элементов, где n - число*

исходящих от блока сигналов. А каждый элемент кортежа может быть либо 1, либо 0, либо другим разбиением.

Опредление: Нулевое разбиение *Под нулевым разбиением понимаем разбиение с пустым параметром fork и кортежем splitting из одного элемента т.е. (, (1))*

Далее введём некоторый набор операций, доступный над разбиениями.

- Произведение двух разбиений задаётся рекурсивной функцией mult от двух аргументов, возвращающей новое разбиение. table 1
- Сложение двух разбиений задаётся рекурсивной функцией sum от двух аргументов, возвращающей новое разбиение. table 2
- Проверка на конкуренцию задаётся рекурсивной функцией concurrent от двух аргументов, возвращающей True в случае выявления конкуренции и False в противном случае, . table 3

В каких случаях используются данные операции над разбиениями будет определено в дальнейшем. А пока приведём несколько примеров их применения:

Умножение двух разбиений:

$$(,(1)) \times (b1,(0,1)) = (,(b1,(1,0)))$$

$$(,(b1,(1,0,1))) \times (b2,(0,1)) = (,(b1,((b2,(0,1)),0,(b2,(0,1)))))$$

Сложение двух разбиений:

$$(,(b1,(1,0))) + (,(b1,(0,1))) = (,(b1,(1,1)))$$

При сложении двух разбиений так же применяется рекурсивное правило *упрощения*, которое определяется как:
Пусть дано разбиение WS = (fork, split), у которого кортеж split состоит из одних единиц и WS не является нулевым разбиением, тогда $WS \equiv 1$:

$$(,(b1,(1,0))) + (,(b1,(0,1))) = (,(b1,(1,1))) \equiv (,(1))$$

Проверка двух разбиений на конкуренцию:

	WS1	1	0
WS2	(WS1.fork, ([(WS1.split[i] × WS2) for i in [0,...,n]]))	WS2	0

Таблица 1: Таблица функции умножения двух разбиений

	WS1	1	0
WS2	if WS1.fork == WS2.fork then (WS1.fork, ([(WS1.split[i] + WS2.split[i]) for i in [0,...,n]])) else throw()	throw()	WS2
1	throw()	throw()	1
0	WS1	1	throw()

Таблица 2: Таблица функции сложения двух разбиений

$$(, (b1, (1, 0))) \perp (, (b1, (0, 1))) = True$$

$$(, (1)) \perp (, (b1, (0, 1))) = False$$

Сравнение на конкуренцию двух разбиений производится следующим образом:

	WS1	1	0
WS2	if WS1.fork == WS2.fork then all([(WS1.split[i] ⊥ WS2.split[i]) for i in [0,...,n]]) else False	False	True
1	False	False	True
0	True	True	False

Таблица 3: Таблица функции проверки двух разбиений на конкуренцию

Так же дадим определение волны, которое будет представлять некоторую связь с сигналом на выходящем порту и содержать информацию о распространяемом потоке внутри workflow. **Определение: Волна(Wave)** *Волной внутри workflow, заданного графом*

связей WFG = (A,D), будем называть связку вида (d, WS) , где $d \in D$ - ребро графа связей, WS - некоторое разбиение.

Для моделирования работы workflow введём такое понятие как состояние workflow. Состояние характеризуется тем моментом, когда группа непоследовательных блоков совершила один шаг, т.е. блоки перешли в новое состояние,

Определение: состояние workflow (Workflow State) *Для worklow с заданным графом связей WFG = (A,D), состояние определяется как набор (BlockStates, WaveFront, BlockHistory)*

- *BlocksStates - отображение $BS : a \to s_a, a \in A, s_a \in states(a)$, сопоставляющая каждому блоку в составе workflow его текущее состояние.*

- *WaveFront - набор волн.*

- *BlockHistory - отображение $H : a \times s_a \times p_{in} \to splits, a \in A, s_a \in states(a), p_{in} \in in(a)$, сопоставляющая каждому блоку, в состоянии s_a и входным портом p_{in} набор разбиений.*

При этом идентификатором уникальности любого состояния workflow WFS = (bs, front, hist) будет только двойка (bs, edges), где edges - набор рёбер $\{d : (d, ws) \in front\}$.

Определение: начальное состояние workflow (Initial Workflow State) *Начальным состоянием workflow будем называть то состояние textit($bs_{init}, front_{init}, hist_{init}$), в котором каждый блок находится в начальном состоянии, в соответствии с его конечным автоматом, и набор волн $front_{init}$, соответствующий испущенным сигналам из выходных портов блока Source. $hist_{init}$ считаем пустым.*

Описание алгоритма нахождения новых состояний workflow за 1 шаг *Пусть дано задано workflow WFS = (bs, front, hist)*

1. *По WaveFront определяем на какие его входные порты поступили сигналы и по его состоянию из BlockStates определяем набор вариантов работы этого блока. Прямое произведение вариантов работы всех блоков на для данного состояния*

2. *При поглощении нескольких сигналов, промежуточное значение разбиения волны*

Функция запуска блока с указанными параметрами evolvestate(WFS, V)

Входные параметры:

WFS = (bs, front, hist) - состояние workflow

V = (a, in, out, state) - вариант запуска блока, где a - идентификатор блока, in - набор входных блока, сигналы по которым он поглотит, в результате работы , out - набор выходных портов блока сигналы по которым будут испущены, state - состояние в которое он перейдёт.

$waves_{in}$ *- набор волн из front, связи которых соединение с портами in блока a.*
$edges_{out}$ *- набор связей, выходящих из портов out,* $n_{out} = |edges_{out}|$ *- число связей.*

Параметр разбиения каждой волны из $waves_{in}$ *проверяем на конкурентность с разбиениями прошедшими, через соответствующий порт блока a, в текущем состоянии блока. В случае, если какая либо пара разбиений окажется конкурентной, означает, что обнаружено потенциальное состояние гонки , что мы считаем ошибкой в композиции workflow. При этом мы знаем детализированную информацию о месте возникновения состояния гонки, и о всех пройденных шагах. Формальная запись условия отсутствия состояний гонки:*

$$\forall_{w \in waves_{in}} \forall_{split_{prev} \in hist(a, bs(a), p_{in})} w.split \perp split_{prev} = False \qquad (1)$$

Если ошибок не обнаружено, то рассчитаем промежуточное значение разбиения потока:
$ws_{sum} = \sum_{w \in waves_{in}} w.split)$

Теперь считаем значения выходных волн:
$waves_{out} = \{(edges_{out}[i], ws_{sum} \times (a, 1_{i,n_{out}}) : i \in [1, n_{out}]\}$.

Под $1_{i,n}$ *понимаем кортеж, состоящий из n элементов, все элементы которого кроме i-го равны 0, а i-ий элемент 1.*

bs_{new} *= bs с изменённым значением bs(a) = state.*
$front_{new} = (front\ waves_{in}) \cup waves_{out}$. $hist_{new}$ *= hist с дополненной информацией пройденных через порты разбиениями потока.*

Возвращаемое значение: состояние workflow (bs_{new}*, front, hist)*

Считаем число

Атомарные блоки по своей сути являются пассивными, т.е. входных данных выдают набор выходных данных на

Связи однозначно определяют траектории распространения сигнала между

5 СЕТИ

5.1 Сети Петри

Сети Петри широко используются для моделирования и исследования динамических дискретных систем. И прежде чем рассмотреть рассмотреть частные случаи использования Сетей Петри, приведём описание их каноничной формы, согласно определению Петерсона [Pet81].

Сеть Петри представляет собой двудольный ориентированный граф, состоящий из вершин двух типов — *позиций* и *переходов*, соединённых между собой дугами. Вершины одного типа не могут быть соединены непосредственно. В позициях могут размещаться метки (маркеры), способные перемещаться по сети.

Определение: Сеть Петри *Простой сетью Петри называется набор* $PN = (S, T, F)$ *, где*

1. $S = \{s_1, \ldots, s_n\}$ - *множество позиций*
2. $T = \{t_1, \ldots, t_r\}$ - *множество переходов таких, что* $S \bigcap T = \varnothing$.
3. $F \subseteq \mu S \times T \times \mu S$ - *отношение инцидентности такое, что*
   - $\forall \langle Q_1^{'}, t_1, Q_1^{''} \rangle, \langle Q_2^{'}, t_1, Q_2^{''} \rangle \in F : \langle Q_1^{'}, t_1, Q_1^{''} \rangle \neq \langle Q_2^{''}, t_2, Q_2^{'} \rangle \Rightarrow t_1 \neq t_2$;
   - $\{t | \langle Q_1^{'}, t_1, Q_1^{''} \rangle \in F\} = T$

Условия в пункте 3 говорят , что для каждого перехода $t \in T$ существует единственный элемент $\langle Q^{'}, t, Q^{''} \rangle$, задающий для него входное множество $Q^{'}$ и выходное множество $Q^{''}$. Дадим определение входному и выходному множеству.

Определение: Входное и выходное множества мест и переходов

Пусть задана сеть $N = (S, T, F)$.

1. *Если для некоторого перехода t имеем* $\langle Q^{'}, t, Q^{''}\rangle \in F$ *, то будем обозначать*
   $\bullet t = Q^{'} = \langle s | t \in s\bullet\rangle, t\bullet = Q^{''} = \langle s | t \in \bullet s\rangle$

2. *И соответственно*
   $\bullet s = \langle t | s \in t\bullet\rangle, s\bullet = \langle t | s \in \bullet t\rangle$

Будем говорить, что $\bullet t$ *- входные , а* $t\bullet$ *- выходные позиции для перехода t. Таким образом, согласно определению, справедливо:* $\forall t \in T : \langle \bullet t, t, t\bullet\rangle \in F$.
Позиция s называться инцидентной переходу t , если $s \in \bullet t$ *или* $s \in t\bullet$.

Сети Петри имеют удобную графическую форму представления в виде графа, в котором места изображаются кружками, а переходы прямоугольниками. Места и переходы, причем место s соединяется с переходом t если $s \in \bullet t$ и t соединяется с s если $s \in t\bullet$.

Само по себе понятие сети имеет статическую природу. Для задания динамических характеристик используется понятие маркировки сети $M \in \mu S$, т.е. функции $M : S \longrightarrow N_0$, сопоставляющей каждому месту целое число. Графически маркировка изображается в виде точек, называемых метками (tokens), и располагающихся в кружках, соответствующих местам сети. Отсутствие меток в некотором месте говорит о нулевой маркировке этого места.

Определение: Маркированная сеть Петри *Маркированной сетью Петри называется набор* (PN, M_0)*, где*

1. *PN =(S,T,F) - сеть;*

2. $M_0 \in \mu S$ *- начальная маркировка.*

Сети Петри были разработаны и используются для моделирования параллельных и асинхронных систем. При моделировании в сетях Петри места символизируют какое-либо состояние системы, а переход символизируют какие-то действия, происходящие в системе. Система, находясь в каком-то состоянии, может порождать определенные действия, и наоборот, выполнение какого-то действия переводит систему из одного состояния в другое.

Текущее состояние системы определяет маркировка сети Петри, т.е. расположение меток (токенов) в местах сети, т.е. $M \in S \to N$. Под маркировкой, представленной в виде: $1s_1 + 2s_2 + 1s_3 + 0s_4$, понимаем, что в позиции s_1 находится 1 метка, 2 метки в s_2, 1 метка

в s_3 и ни одной метки в s_4. Так же представление этой маркировки можно сократить до $1s_1+2s_2+1s_3$. Для сравнения двух маркировок, мы определяем частичный порядок.

$$\forall_{M_1,M_2} M_1 \leq M_2, \forall_{s\in S} : M_1(s) \leq M_2(s).$$

Выполнение действия в системе, в сетях Петри определяется как срабатывание переходов. Срабатывание переходов порождает новую маркировку, т.е. порождает новое размещение меток (токенов) в сети.

Работа мариковочной сети Петри управляется наличием или отсутствием маркировочных токенов. В сети Петри срабатывает переход t, в процессе которого с каждого входа $s \in \bullet t$ снимается метка и каждому выходу $s \in t\bullet$ прибавляется метка.

Определение: Правило срабатывания переходов *Пусть* $\Sigma = (S, T, F, M_0)$ *маркировочная сеть.*

1. *Переход* $t \in T$ *считается возбуждённым при маркировке* $M \in \mu S$*, если каждое положение* $s \in \bullet t$ *имеет хотя бы одна метка;*

2. *Переход* t*, возбуждённый при маркировке* M*, может сработать, приведя к новой маркировке* $M^{'}$*, которая получается путём поглощения метки на каждом позиции* $s \in \bullet t$ *и появления новой метки на каждой позиции* $s' \in t\bullet$*.*

Для сети Петри (S, T, F) и её текущей маркировки $M_1 \in \mu S$ опишем следующие обозначения:

- $M_1 \xrightarrow{t} M_2$: преход t при маркировке M_1 возбуждён и при его срабатывание приводит к маркировке M_2

- $M_1 \rightarrow M_2$: существует переход t, такой что $M_1 \xrightarrow{t} M_2$

- $M_1 \xrightarrow{\sigma} M_n$: последовательность переходов $\sigma = t_1t_2t_3...t_{n-1} \in T^*$ переводящая маркировку M_1 в M_n через набор (возможно пустой) промежуточных маркировок $M_2, ..., M_{n-1}$, т.е.: $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_1} ... \xrightarrow{t_{n-1}} M_n$
  Маркировка M_n называется *достижимой* из M_1 (и имеет обозначение $M_1 \xrightarrow{*} M_n$) тогда и только тогда, когда существует последовательность такая переходов σ, что $M_1 \xrightarrow{\sigma} M_n$. Отметим, что пустая последовательность переходов тоже допустима,

$M_1 \xrightarrow{*} M_1$.

Когда же мы имеем сеть Петри с начальной маркировкой (S, T, F, M_0), то маркировка M' будет допустимой, если $M_0 \xrightarrow{*} M'$.

Определение: Живость(Live) *Маркированная сеть Петри (PN, M) называется живой, если для любой достижимой маркировки M' и любого перехода t существует маркировка M'' достижимая из маркировки M' и задействующая переход t.*

Определение: Ограниченность(Bounded), безопасность(Safe) *Маркированная сеть Петри (PN, M) называется ограниченной, если для каждого положения s существует такое натуральное число n, что для каждой достижимой маркировки меток в этом положении меньше n. Сеть называется безопасной, если максимальное число меток не превышает 1.*

Определение: Правильность(Well-formed) *Сеть Петри PN называется правильной, если существует такая начальная маркировка M_0, что сеть(PN, M_0) будет живой и ограниченной.*

Определение: Путь *Пусть дана сеть Петри PN. Путь C из вершины n_1 в вершину n_k представляет собой последовательность вершин $\langle n_1, n_2, ..., n_k \rangle$ такую, что $\langle n_i, n_{i+1}, n_{i+2} \rangle \in F$ для $1 \leq i \leq k-2$.*

Путь C называется простым, если для любых двух вершин $n_i, n_j \in C$ выполняется: $i \neq j \Rightarrow n_i \neq n_j$.

Путь C называется бесконфликтным, если для любой положения n_j и любого перехода n_i , $n_i, n_j \in C$, выполняется $j \neq i-1 \Rightarrow n_j \notin {\bullet}n_i$.

Для удобства введём оператор α над путями. Тогда для пути $C = \langle n_1, n_2, ..., n_k \rangle$ $\alpha(C) = \{n_1, n_2, ..., n_k\}$

Определение: Сильная связность(Strongly Connected) *Сеть Петри называется сильно связной, если для любой пары вершин x и y существует путь из x в y.*

Определение: Свободный выбор(Free-Choice) *Сеть Петри обладает свойством свободного выбора, если для любых двух переходов $t_1, t_2 \in T$ выполняется ${\bullet}t_1 \cap {\bullet}t_2 \neq \varnothing \Rightarrow {\bullet}t_1 = {\bullet}t_2$*

ПРИМЕРЫ

Определение: WF-сеть(Workflow-net) *Пусть дана сеть Петри PN = (S,T,F).*

- Если PN - WF-сеть с входной позицией i, тогда для любой пози-

ции $s \in S : \bullet s \neq \{s = i$, *т.е.* i *-единственная входная позиция.*

- *Если PN - WF-сеть с выходной позицией* o*, тогда для любой позиции* $s \in S : s\bullet \neq \{s = o$, *т.е.* o *-единственная выходная позиция.*

- *Если PN - WF-сеть и мы добавляем в PN переход* t^**, соединяющий выходную позицию* o *и входную позицию* i *(т.е.* $\bullet t^* = o, t^*\bullet = i$*), тогда получившаяся сеть будет обладать сильной связностью.*

Для WF-сети начальную маркировку с единственной меткой только во входной позиции i *будем обозначать* $_i$*, а маркировку с единственной меткой только в выходной позиции* o *соответственно* $_o$.

Определение: Корректность(Soundness) *WF-сеть PN = (S,T,F) корректна тогда и только тогда, когда:*

(i) Для каждой маркировки M *,достижимой из* i*, маркировка* o *так же достижима из* M.

$$\forall M(i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$$

(ii) Маркировка o *- единственная достижимая из* i *маркировка, имеющая хотя бы одну метку в позиции* o.

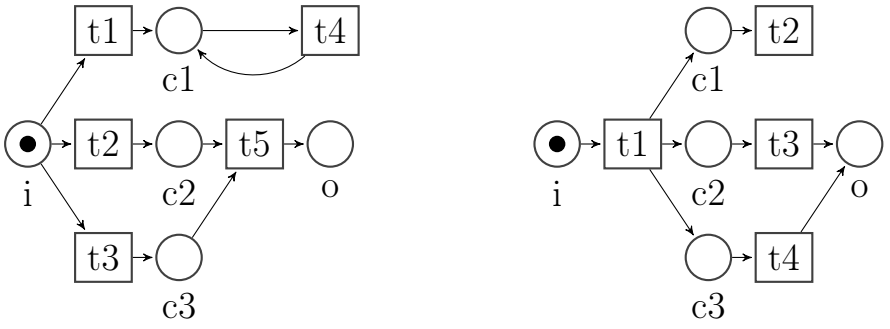
$$\forall_M(i \xrightarrow{*} M \wedge M \geq o) \Rightarrow (M = o)$$

(iii) В (PN,i) нет неживых переходов.

$$\forall_{t \in T} \exists_{M,M'} i \xrightarrow{*} M \xrightarrow{t} M'$$

Рассмотрим примеры нарушения условий корректности WF-сетей.

- Для WF-сети , изображённой на Рис. 1(a) нарушено первое условие корректности. Срабатывание любого из переходов t1,t2 или t3 приводит к маркировке, из которой маркировка o недостижима. При маркировке c1 мы имеем бесконечный цикл,а в случае маркировки c2 или c3 переход t5 никогда не сработает, т.к. требует по метке на каждой из входных позиций.



(a) (b)

Рис. 1: This is a figure.

- В результате выполнения последовательности переходов $\langle t1, t3, t4\rangle$ WF-сети Рис. 1(b) нарушается второе условие корректности,

- В сети , изображённой на Рис. 1(b), так же нарушено третье условие корректности, т.к. переход t2 - неживой.

Для данной WF-сети PN = (S,T,F) мы хотим определить, является ли она корректной. В [2] показано, как свойство корректности соотносится с живостью и ограниченностью сети. Для того, чтобы связать эти понятия,дадим определение расширенной сети $\overline{PN} = (\overline{S}, \overline{T}, \overline{F})$, где $\overline{S} = S, \overline{T} = T \cup \{t^*\}, \overline{F} = F \cup \langle o, t^*\rangle, \langle t^*, i\rangle\}$.

Теорема 1. *WF-сеть PN корректна тогда и только тогда, когда $(\overline{PN}, i)$ - живая и ограниченная маркировочная сеть.*

5.2 Струтурные характеристики корректности

Корректности WF-сети является её динамической характеристикой и из это вытекают некоторые проблемы:

- Для сложных WF-сетей задача определения корректности может быть весьма затратной (Для произвольных WF-сетей вычисление ограниченности и живости имеет экспоненциальную сложность[8]).

- Теорема 1 не определяет в какие именно компоненты WF-сети нарушают свойство корректности.

Поэтому полезно было бы знать какими структурными характеристиками обладают корректные WF-сети. Для этого будут рассмотрены некоторые классы WF-сетей: WF-сети со свободным выбором, хорошо структурированные WF-сети/

5.2.1 WF-сети со свободным выбором

Напомним определение свободного выбора в сети. Если для любых двух переходов $t_1, t_2 \in T$, для которых выполняется $\bullet t_1 \cap \bullet t_2 \neq \varnothing$, то необходимо чтобы $\bullet t_1 = \bullet t_2$, т.е. эти переходы являются частью одного OR-split.
Соблюдение правила свободного выбора не мешает моделировать параллельное,последовательное и циклическое выполнение задач.
Если же мы будем допускать сети не обладающие свойством свободного выбора,то тогда на выбор между конкурирующими задачами влияет очерёдность в которой выполнялись предшествующие задачи, что является недопустимым при грамотном проектировании. Нарушение свободы выбора,чаще всего возникает при сочетании распараллливания задач и маршрутизации с выбором.

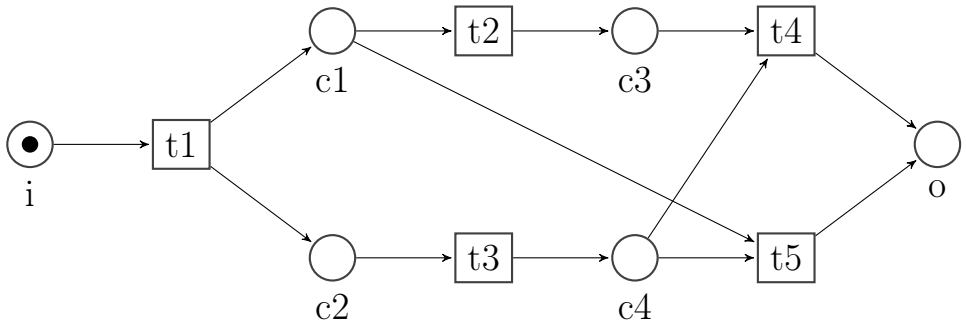


Рис. 2: This is a figure.

На Рис. 2 изображена такая ситуация. Сработавший переход t1 вводит параллелизм. Но тем не менее, в выбор между t2 и t5 ,т.к. переход t5 не возбуждён. Параллельное выполнение t2 и t3 приводит к ситуации, когда выполнить переход t5 невозможно. Тем не менее, если выполнение перехода t2 отложено до окончания выполнения t3, то тогда будет существовать выбор между t2 и t5. Но хотелось бы , чтобы параллелизм был отделён от выбора между альтернативами. Поэтому мы считаем конструкцию, приведённую на Рис. 2, неверной.

Вывод 1. *Для WF-сети со свободным выбором, задача проверки на корректность решается за полиномиальное время.*

Док-во. Пусть PN - WF-сеть со свободным выбором, тогда $\overline{PN}$ тоже будет обладать свободным выбором. В работе [10] доказано, что определение живости и ограниченности для($\overline{PN}$), можно за полиномиальное время. По Теорема 1 это эквивалентно корректности WF-сети.

Лемма 1. *Корректная WF-сеть со свободным выбором - безопасная.*

Док-во. Пусть PN - корректная WF-сеть со свободным выбором. Тогда $\overline{PN}$ тоже будет со свободным выбором и правильностью(well-formed). Следовательно, $\overline{PN}$ S-покрываема[10], т.е. каждая позиция в . И т.к. в начальной маркировке есть всего одна метка,то ($\overline{PN}$,i) будет безопасной,а следовательно и (PN,i).

Безопасность является предпочтительным свойством, т.к. нелогично было бы допускать несколько меток в позиции, представляющей состояние. Состояние может быть активным (1 метка) или неактивным (ни одной метки). Non- free-choice constructs such as the construct shown in Figure 4 are a potential source of anomalous behavior (e.g., deadlock) which is difficult to trace.

5.2.2 Хорошо структурированные WF-сети

Другой подход для определения хорошей структурной характеристики workflow - это сбалансированность компонент типов AND/OR-split и AND/OR-join. Очевидно, что два параллельных потока инициированных AND-split, не должны объединяться через OR-join.Так же как и два альтернативных потока , инициированных компонентом OR-split не должны синхронизироваться через AND-join.

Для того чтобы формализовать конструкции на рисунке Рис. 3 дадим следующее определение.

Хорошая организованность(Well-handled) *Сеть Петри PN называется хорошо организованной, если для любых пар вершин x и y таких , что одна из вершин является позицией , а другая переходом и для любой пары простых путей C_1 и C_2 из x в y, $\alpha(C_1 \cup \alpha(C_2) = \{x, y\} \Rightarrow C_1 = C_2$.*

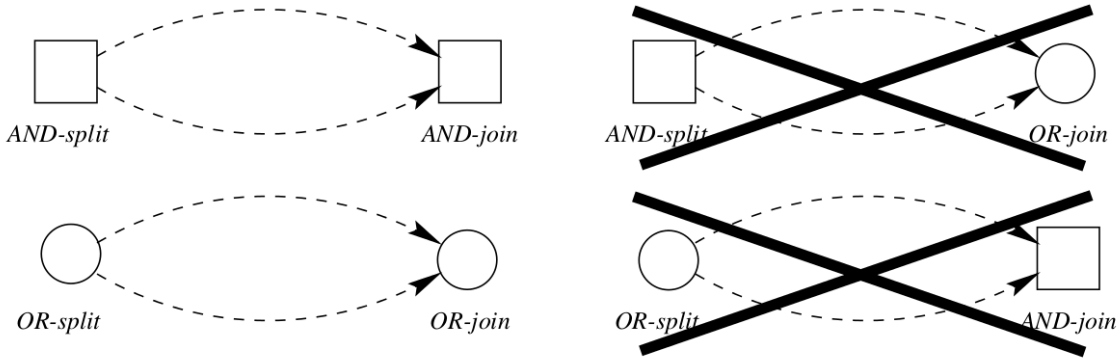


Рис. 3: Хорошие и плохие конструкции

Хорошая организованность может быть определена за полиномиальное время применяя алгоритмы нахождения максимального потока и минимального среза , описанные в [5].

Лемма 2. *Хорошая организованность сеть петри с сильной связностью будет хорошо организованной.*

Доказательство. Пусть PN - хорошая организованность сеть петри с сильной связностью. Очевидно, что не будет существовать ни одно пути

Хорошая структурированность(Well-structured) *WF-сеть является хорошая структурированной, если* $\overline{PN}$ *хорошая организованная.*

Список литературы

[1] Gray L., Griffeath D. The ergodic theory of traffic jams // J. Stat.

[2] Y. Han, A. Sheth, C. Bussler, A Taxonomy of Adaptive Workflow Management, in: Conference on Computer–Supported Cooperative Work (CSCW-98), Seattle, WA, 1998. URL http://ccs.mit.edu/klein/cscw98/

10