ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 7
N 8
2016

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4′42 DOI: 10.17587/prin.7.339-350

А. В. Пилипенко, аспирант, e-mail: artur.pilipenko@gmail.com, СПбГУ, г. Санкт-Петербург, Россия,
О. А. Плисс, канд. физ.- мат.наук, Principal Member of Technical Staff, e-mail: oleg.pliss@gmail.com, Oralce, Санта-Клара, США

Понижение избыточности Java-программ при выборочной инициализации классов

Рассмотрена задача удаления неиспользуемых методов, полей и классов Java-программ при создании загрузочного образа инициализированного состояния виртуальной машины. Один из способов оптимизации при подготовке такого образа — это инициализация классов. Созданные при этом объекты могут влиять на достижимость методов и удалимость полей и классов. Предложен алгоритм анализа достижимости методов, осуществляющий выборочную инициализацию классов программы. Также предложены алгоритмы анализа удалимости полей и классов, которые учитывают зависимости между методами, полями, классами и объектами, созданными при инициализации классов.

Ключевые слова: *Java, виртуальные машины, инициализация классов, анализ достижимости, косвенные вызовы*

Введение

При каждом старте и инициализации виртуальной машины выполняются одни и те же действия. Создаются системные объекты, загружаются классы стандартных библиотек, разрешаются символические ссылки, инициализируются загруженные классы. Этот процесс можно оптимизировать, если сохранить состояние инициализированной виртуальной машины в виде загрузочного образа. Тогда при ее старте и инициализации достаточно будет восстановить состояние из сохраненного образа. Подготовка образа осуществляется специальной версией виртуальной машины. При этом кроме стандартной инициализации могут осуществляться различные оптимизации, например, удаление неиспользуемых методов, полей и классов. Такой подход часто используется для оптимизации виртуальных машин, предназначенных для работы на устройствах с ограниченными ресурсами [2, 9, 10].

Инициализация классов при подготовке образа сокращает время инициализации приложения, ускоряет исполнение и во многих случаях позволяет уменьшить размер образа. Удаление неиспользуемых методов, полей и классов уменьшает статическое и динамическое потребление памяти. Такой способ оптимизации особенно эффективен в закрытой модели, когда весь код, исполняемый виртуальной машиной, известен заранее и доступен для статического анализа.

Классический подход к удалению недостижимого кода состоит в построении транзитивного замыкания методов, достижимых из точек входа программы, и удаления методов, не попавших в замыкание. Инициализация классов влияет на достижимость методов. Во-первых, статические инициализаторы инициализированных классов становятся недостижимы. Во-вторых, объекты, созданные при инициализации классов, могут быть использованы для вызовов виртуальных и интерфейсных методов. Классические алгоритмы построения транзитивного замыкания не учитывают эти зависимости.

Объекты, созданные при инициализации классов, также влияют на удалимость полей. Удаление полей может изменить достижимость объектов, что способно привести к нарушению поведения программы.

Авторами предложен алгоритм анализа достижимости методов, осуществляющий выборочную инициализацию используемых классов. Данный алгоритм определяет, какие из объектов, созданных во время инициализации классов, будут доступны для косвенных вызовов. Предложенный алгоритм является развитием алгоритма анализа достижимости методов, описанного авторами в статье [1]. Также предложены алгоритмы анализа удалимости полей и классов, которые учитывают объекты, созданные при инициализации классов.

В разд. 1 описаны классические алгоритмы построения транзитивного замыкания достижимых методов. В разд. 2 описана инициализация классов при подготовке образа. В разд. 3 обсуждена достижимость объектов в куче. Разд. 4 посвящен описанию предложенного алгоритма анализа достижимости методов. В разд. 5 и 6 описаны предложенные авторами алгоритмы анализа удалимости полей и классов. Эффективность практического применения алгоритмов проанализирована в разд. 7. В разд. 8 представлен обзор смежных работ. Формальное описание предложенных алгоритмов приведено в Приложении.

1. Транзитивное замыкание графа вызовов

Если метод *f* может быть вызван в результате выполнения метода *g*, метод *f* считается достижимым из метода *g*. Для нахождения используемых методов строится транзитивное замыкание методов, достижимых из точек входа программы.

Для нахождения методов, достижимых из заданного, анализируются инструкции вызова в коде метода. Аргумент инструкции прямого вызова (`invokestatic`, `invokespecial`) статически определяет вызываемый метод. Метод, вызываемый инструкцией косвенного вызова (`invokevirtual`, `invokeinterface`)[1], определяется на этапе исполнения. Для динамического связывания используется объект-получатель — при выполнении инструкции вызывается реализация метода-аргумента в классе данного объекта.

При статическом анализе отсутствует информация об объектах времени исполнения. Для анализа достижимости используется консервативное приближение множества классов объектов-получателей. В простом случае можно считать, что для косвенного вызова может быть использован объект любого типа, совместимого со статическим типом вызова. То есть множество возможных классов — это множество всех подклассов класса, в котором определен метод-аргумент. Здесь и далее будем считать, что сам класс принадлежит множеству его подклассов. Описанный подход называется Class Hierarchy Analysis (CHA) [4].

Точность анализа можно повысить, если учесть, что нестатический метод класса может быть вызван только тогда, когда существуют или могут быть созданы объекты данного класса или его подклассов. Уточненный алгоритм называется Rapid Type Analysis (RTA) [3]. В процессе работы алгоритма вычисляются множество косвенных вызовов и множество классов, экземпляры которых создаются достижимым кодом. Обозначим эти множества indirect_invocations и instantiable_classes соответственно. Итеративный алгоритм начинается с добавления в замыкание точек входа приложения. Алгоритм завершается по достижению неподвижной точки. На каждой итерации выполняются следующие действия.

1. Для каждого ранее не обработанного метода из замыкания просматривается его код:

- методы, вызываемые инструкциями прямого вызова, добавляются в замыкание;
- косвенные вызовы добавляются в множество indirect_invocations;
- классы, используемые для создания объектов с помощью инструкции `new`, добавляются в множество instantiable_classes.

2. Для каждого вызова из множества indirect_invocations вычисляется пересечение множества подклассов статического класса вызова с множеством instantiable_classes. Полученное пересечение — это классы, методы которых могут быть вызваны. Соответствующие методы добавляются в замыкание.

При построении замыкания методы `<clinit>` и `finalize` требуют специальной обработки. Метод `<clinit>` выполняется при инициализации класса и не может быть вызван стандартными инструкциями. Обработка методов `<clinit>` описана далее в разд. 2 и 4.

Метод `finalize` определен в классе `Object` и может быть переопределен в подклассах. Финализатором объекта называется реализация метода `finalize` в классе объекта. Финализатор выполняется при уничтожении объекта сборщиком мусора. Кроме того, он может быть вызван явно. Класс `Object` содержит пустую реализацию метода `finalize`. Объект с непустым финализатором будем называть финализируемым. Обработка финализируемых объектов подробно рассмотрена в разд. 3—5.

2. Инициализация классов

Инициализация классов в Java осуществляется при первом обращении к классу, либо при инициализации одного из его подклассов. Обращением к классу считается:

- выполнение одной из инструкций `new`, `invokestatic`, `getstatic`, `putstatic`;
- рефлексивный доступ к классу, например, с помощью метода `Class.forName`.

В процессе инициализации вызывается статический инициализатор класса — метод `<clinit>` [6]. Так как данный метод недоступен для вызова стандартными инструкциями, после инициализации класса он становится недостижим.

Ранняя инициализация классов при создании образа ускоряет исполнение приложения. Во-первых, сокращается время инициализации. Во-вторых, инструкции `new`, `invokestatic`, `getstatic`, `putstatic`, которые ссылаются на инициализированные классы, могут быть заменены быстрыми версиями, которые не проверяют, инициализирован класс или нет. Такая оптимизация невозможна на этапе исполнения, если код методов размещается в неизменяемой памяти.

В большинстве случаев ранняя инициализация классов сокращает размер образа. Инициализация классов уменьшает набор достижимых методов, но в то же время может увеличить число объектов в образе за счет объектов, созданных инициализаторами. Для большинства классов вклад созданных объектов в размер образа оказывается меньше, чем сокращение размера за счет уменьшения числа достижимых методов.

Не все классы можно инициализировать при создании образа. Результат работы инициализатора может иметь побочные эффекты, например, инициализатор может осуществлять операции ввода—вывода. Результат работы инициализатора может зависеть от того, на какой виртуальной машине он вызван. Например, инициализатор может использовать метод `System.getProperty()`, результат работы которого

[1] Набор инструкций в Java 7 был расширен инструкцией вызова `invokedynamic` для поддержки динамических языков программирования. Для анализа нового механизма диспетчеризации описываемый алгоритм нуждается в дополнении

отличается на этапе создания образа и на этапе исполнения. В общем случае ранняя инициализация классов нарушает семантику языка и может изменить видимое поведение программы.

Класс не будет инициализирован при создании образа, если в его инициализаторе присутствует хотя бы одна из перечисленных далее операций.

- Цикл. Инициализатор с циклами может никогда не завершиться.
- Вызов методов. В частности, запрещен вызов конструкторов объектов. Запрет вызова методов позволяет не анализировать код вызываемых методов.
- Запись в статическое поле класса, отличного от данного класса и его надклассов. Эта операция может также привести к нежелательной инициализации класса.

Данная эвристика может оказаться слишком консервативной. Запрет на инициализацию класса можно отменить при помощи аннотации `@InitAtBuild`.

Следует отметить, что при выполнении инициализатора может быть брошено исключение. При ленивой инициализации классов это исключение может быть обработано кодом приложения. При создании образа такое исключение нельзя обработать, поэтому при возникновении исключительной ситуации создание образа завершается ошибкой. Поэтому для некоторых корректных с точки зрения языка приложений создать загрузочный образ невозможно.

Наиболее эффективно инициализировать классы в процессе построения замыкания. До построения замыкания невозможно определить, какие классы могут быть инициализированы достижимым кодом. Инициализация лишних классов может создать в памяти достижимые, но не используемые объекты. Инициализация классов после построения замыкания менее эффективна, так как после удаления инициализаторов некоторые ранее достижимые методы могут утратить это свойство.

Каждый потенциально инициализируемый достижимым кодом класс можно либо инициализировать на этапе построения замыкания, либо отложить его инициализацию до этапа исполнения. В первом случае метод `<clinit>` должен быть исполнен, во втором случае `<clinit>` и все достижимые из него методы должны быть добавлены в замыкание.

3. Достижимость объектов в куче

В результате выполнения инициализаторов могут быть созданы объекты, некоторые из которых останутся достижимыми для приложения. Эти объекты могут быть использованы для косвенных вызовов. В простом случае можно считать, что для приложения достижимы все объекты, пережившие сборку мусора после инициализации классов. Однако этот метод неточен — сборщик мусора проходит по всем ссылочным полям, не принимая во внимание их доступность приложению. Для точного анализа необходимо самостоятельно построить множества читаемых полей и достижимых объектов.

В коде достижимых методов проанализируем инструкции чтения полей (`getstatic`, `getfield`). Поля, используемые этими инструкциями, добавим в множество читаемых полей.

Будем считать достижимыми для приложения объекты, достижимые из корневых ссылок по ссылкам в читаемых полях, и финализируемые объекты. Финализируемые объекты достижимы через ссылку `this` при выполнении метода `finalize` вне зависимости от достижимости по ссылкам в читаемых полях. При этом специальные ссылки (`WeakReference`, `SoftReference`, `PhantomReference`) обрабатываются наравне с обычными ссылками.

4. Анализ достижимости методов

Дополним алгоритм RTA инициализацией классов. В процессе работы алгоритма будем вычислять следующие множества:

- reachable_methods, new_reachable_methods — множество достижимых методов и его рабочее подмножество соответственно;
- indirect_invocations, new_indirect_invocations — множество методов, используемых для косвенных вызовов, и его рабочее подмножество соответственно;
- instantiable_classes, new_instantiable_classes — множество классов, экземпляры которых доступны для косвенных вызовов, и его рабочее подмножество соответственно;
- initializable_classes, new_initializable_classes — множество классов, потенциально инициализируемых достижимым кодом, и его рабочее подмножество соответственно;
- read_fields — множество читаемых полей;
- written_object_fields — множество объектных полей, для которых существуют операции записи;
- putstatic_fields — множество статических полей, для которых существуют операции записи, вероятно приводящие к инициализации класса;
- referenced_classes — множество используемых классов.

Множества written_object_fields, putstatic_fields и referenced_classes впоследствии будут использованы для анализа удалимости полей и классов.

При добавлении нового элемента в множество, у которого есть рабочее подмножество, будем добавлять элемент в множество и соответствующее рабочее подмножество.

Итеративный алгоритм начинается с добавления точек входа приложения в множество reachable_methods. Алгоритм завершается, когда множества new_reachable_methods и new_initializable_classes пусты. На каждой итерации алгоритма выполняются следующие шаги.

Шаг 1. Для каждого метода из множества new_reachable_methods просматриваем его код:

- методы, вызываемые инструкциями прямого вызова, добавляем в множество reachable_methods;
- методы, используемые инструкциями `invokevirtual` и `invokeinterface`, запоминаем в множестве indirect_invocations;

- классы, используемые для создания объектов с помощью инструкции `new`, запоминаем в множестве instantiable_classes;
- классы, на которые ссылаются инструкции `new`, `invokestatic`, `getstatic`, `putstatic`, добавляем в множество initializable_classes;
- поля, используемые инструкциями `getstatic`, `getfield`, добавляем в множество read_fields;
- поля, используемые инструкцией `putstatic` добавляем в множество putstatic_fields, если анализируемый метод не принадлежит подклассу класса, содержащего поле;
- объектные поля, используемые инструкциями `putstatic`, `putfield`, добавляем в множество written_object_fields;
- классы, на которые ссылаются инструкции `ldc`, добавляем в множества instantiable_classes и initializable_classes;
- классы, на которые ссылаются инструкции `new`, `invokespecial`, `invokestatic`, `invokevirtual`, `invokeinterface`, `getstatic`, `putstatic`, `getfield`, `putfield`, `ldc`, `instanceof`, `checkcast`, добавляем в множество referenced_classes.

Классы исключений из таблицы обработчиков исключений данного метода добавляем в множество referenced_classes.

Проанализированные методы удаляем из множества new_reachable_methods.

Шаг 2. Обрабатываем классы из множества new_initializable_classes. Инициализируем классы, которые могут быть инициализированы согласно эвристике из разд. 2. Инициализаторы остальных классов добавляем в множество reachable_methods. Обработанные классы удаляем из множества new_initializable_classes.

Шаг 3. Выполняем сборку мусора, финализируем удаленные объекты.

Шаг 4. Обходим объекты, достижимые для приложения. Для этого:

- обходим все финализируемые объекты,
- обходим граф объектов, достижимых из корневых ссылок по обычным и специальным ссылкам. По ссылке, содержащейся в поле, переходим только если поле принадлежит множеству read_fields.

Классы посещенных объектов добавляем в множество instantiable_classes.

Шаг 5. В множество reachable_methods добавляем методы, доступные через косвенные вызовы:

- добавляем методы классов из множества instantiable_classes, достижимые через косвенные вызовы методов из множества new_indirect_invocations;
- добавляем методы классов из множества new_instantiable_classes, достижимые через косвенные вызовы методов из множества indirect_invocations.

Шаг 6. В множество reachable_methods добавляем финализаторы классов из new_instantiable_classes.

Шаг 7. Обнуляем множества new_instantiable_classes, new_indirect_invocations.

После завершения алгоритма методы, не принадлежащие объединению множеств indirect_invocations и reachable_methods, можно удалить, не нарушив поведения программы. Методы, принадлежащие разности множеств indirect_invocations и reachable_methods, являются эффективно абстрактными. Тела таких методов можно удалить.

Заметим, что описанный алгоритм извлекает статические зависимости из байт-кода программы. Однако не все зависимости можно проанализировать таким образом. Зависимости, возникающие при доступе к сущностям языка Java из нативного кода, не отражены в байт-коде. Использование рефлексивного доступа порождает динамические зависимости, которые невозможно проанализировать на этапе построения образа. Для того чтобы корректно обрабатывать такие ситуации, реализация предложенного алгоритма позволяет вручную описывать дополнительные зависимости. Для каждого метода можно указать какие поля, методы и классы используются этим методом. Такие зависимости анализируются вместе с кодом метода на шаге 1.

5. Анализ удалимости полей

После завершения основного алгоритма вычислим unremovable_fields — множество полей, удаление которых может нарушить видимое поведение программы. Поля, которые можно удалить, не нарушив поведения программы, назовем удалимыми. Считая, что читаемые поля не удалимы, включим в множество unremovable_fields поля из множества read_fields.

Не все нечитаемые поля удалимы. Для некоторых нечитаемых полей в коде достижимых методов могут существовать операции записи. Можно было бы считать, что любая запись в поле (`putstatic`, `putfield`) влияет на видимое поведение. Однако часто оказывается, что инициализированные в конструкторе или статическом инициализаторе поля не используются достижимым кодом.

В языке Java запись в поле может сопровождаться нетривиальными побочными эффектами. Запись в статическое поле (`putstatic`) может вызвать инициализацию содержащего это поле класса. В некоторых случаях можно гарантировать, что класс, содержащий поле, в момент исполнения инструкции уже будет инициализирован, например, если класс был инициализирован в процессе построения замыкания или если анализируемый метод принадлежит подклассу класса, содержащего поле.

Множество putstatic_fields содержит статические поля, для которых существуют операции записи, вероятно приводящие к инициализации класса. Включим в множество unremovable_fields поля неинициализированных классов, содержащиеся в множестве putstatic_fields.

Ссылки в нечитаемых объектных полях препятствуют уничтожению сборщиком мусора объектов, достижимых через эту ссылку. Уничтожение объекта влияет на поведение программы, если объект финализируем или доступен посредством специальной ссылки. Поле может содержать ссылку, если оно

инициализировано ненулевой ссылкой в момент анализа, либо если в коде достижимых методов присутствуют операции записи в это поле. Будем считать, что поле нельзя удалять, если через ссылку в поле может быть достижим финализируемый объект или объект, доступный посредством специальной ссылки. Для проверки достижимости финализируемых объектов применим анализ типов.

Поле может ссылаться на экземпляр класса *A*, если выполнено хотя бы одно из двух условий:

- в памяти существует экземпляр поля, содержащий ссылку на экземпляр класса *A*;
- в процессе выполнения в поле может быть записана ссылка на экземпляр класса *A*, а именно
  - класс *A* совместим с объявленным типом поля;
  - в коде достижимых методов есть операции записи в поле;
  - если поле нестатическое, класс, содержащий поле, принадлежит множеству instantiable_classes.

Экземпляр класса *A* может ссылаться на экземпляр класса *B*, если класс *B* принадлежит множеству instantiable_classes и у класса *A* существует нестатическое поле, которое может ссылаться на экземпляр класса *B*.

Сложнее доказать недостижимость объектов, доступных посредством специальных ссылок. В Java-коде специальные ссылки представлены классами `WeakReference`, `SoftReference`, `PhantomReference`. В исходном коде тип специальной ссылки параметризован классом объекта, однако эта информация стирается после компиляции. Будем считать, что если приложение использует специальные ссылки, то они могут ссылаться на объекты любых инстанциируемых классов.

Дополним множество unremovable_fields полями, через ссылки в которых могут быть достижимы объекты, уничтожение которых приложение может отследить. Если в множестве instantiable_classes присутствует хотя бы один из классов специальных ссылок, дополним множество unremovable_fields объектными полями, содержащими ненулевые ссылки, и полями из множества written_object_fields. В противном случае построим множество полей, которые могут прямо или косвенно ссылаться на экземпляры финализируемых классов.

Для каждого класса предварительно вычислим два описанных далее множества:

- may_refer_to(class) — множество полей, объявленный тип которых позволяет сохранить в поле ссылку на экземпляр класса. Для вычисления переберем все поля, каждое поле добавим в множества may_refer_to(class) подклассов объявленного типа поля.
- refer_to(class) — множество полей, экземпляры которых ссылаются на экземпляры класса. Для вычисления обойдем граф объектов, достижимых из корневых ссылок. Посещенные поля добавим в множество refer_to соответствующих классов.

В процессе работы алгоритма будем вычислять следующие множества:

- ref_to_finalizable_fields, new_ref_to_finalizable_fields — множество полей, которые могут прямо или косвенно ссылаться на финализируемые классы, и его рабочее подмножество соответственно;
- ref_to_finalizable_classes, new_ref_to_finalizable_classes — множество классов, экземпляры которых могут прямо или косвенно ссылаться на финализируемые классы, и его рабочее подмножество соответственно.

При добавлении нового элемента в множество ref_to_finalizable_fields или ref_to_finalizable_classes будем добавлять элемент в множество и соответствующее рабочее подмножество.

Итеративный алгоритм начинается с инициализации множества ref_to_finalizable_classes подмножеством классов из instantiable_classes, у которых определен метод `finalize`. Алгоритм завершается, когда множество new_ref_to_finalizable_classes становится пустым. На каждой итерации выполняются следующие действия.

1. Для каждого класса из множества new_ref_to_finalizable_classes поля, которые могут ссылаться на экземпляр класса, добавляются в множество ref_to_finalizable_fields. Это поля, принадлежащие множеству may_refer_to(class) ∩ (refer_to(class) ∪written_object_fields). Обработанные классы удаляются из множества new_ref_to_finalizable_classes.

2. Для каждого нестатического поля из множества new_ref_to_finalizable_fields классы, экземпляры которых могут содержать поле, добавляются в множество ref_to_finalizable_classes. Это подклассы класса, в котором определено поле, принадлежащее множеству instantiable_classes. Обработанные поля удаляются из множества new_ref_to_finalizable_fields.

По завершению алгоритма дополним множество unremovable_fields полями из множества ref_to_finalizable_fields.

6. Анализ удалимости классов

После анализа достижимости методов и удалимости полей вычислим множество unremovable_classes — множество неудалимых классов.

- Классы, на которые ссылаются инструкции достижимых методов, удалять нельзя, инициализируем множество unremovable_classes классами из множества referenced_classes.
- Класс нельзя удалять, если для приложения достижим экземпляр класса. Такие классы содержатся в множестве instantiable_classes, поэтому дополним множество unremovable_classes классами из множества instantiable_classes.
- Класс нельзя удалять, если он содержит неудалимое поле. Если такое поле было проинициализировано при инициализации классов, то в коде достижимых методов может не оказаться использований этого поля, а класс, содержащий поле, может отсутствовать в множестве referenced_classes. Дополним множество unremovable_classes классами, содержащими неудалимые поля.

Классы, не принадлежащие множеству unremovable_classes, можно удалить, не нарушив поведения программы.

Таблица 1

Размер образа при различных оптимизациях, байт

Конфигурация	Вид оптимизации				
	none	init	eliminate	init + eliminate	
CLDC	308 813	308 245	299 657	298 521	(96,7)
MEEP	1 806 975	1 795 391	1 616 627	1 594 515	(88,2)
CLDC + Hello World	255 476	254 904	12 288	11 184	(4,4)
CLDC + EEMBC	429 108	428 540	189 216	188 376	(43,9)
Примечание: в столбце init + eliminate в скобках указан размер в процентах относительно значения в столбце none.					

Таблица 2

Число классов

Конфигурация	Всего классов	Классы с непустым статическим инициализатором	Классы, потенциально инициализируемые достижимым кодом	Классы, инициализированные предложенным алгоритмом
CLDC	373	42	361	361
MEEP	2194	326	1869	1721
CLDC + Hello World	374	42	53	53
CLDC + EEMBC	499	59	207	185

7. Анализ эффективности

Для анализа эффективности предложенного алгоритма был измерен размер образа нескольких конфигураций при различных оптимизациях. Для измерений были использованы следующие конфигурации:

- CLDC — реализация стандарта Connected Limited Device Configuration 8 (JSR360)[2] в открытой модели;
- MEEP — реализация стандарта Java ME Embedded Profile (JSR361) в открытой модели;
- CLDC + Hello World — минимальное приложение для платформы CLDC в закрытой модели;
- CLDC + EEMBC — набор тестов EEMBC GrinderBench в закрытой модели.

Ниже приведена информация о размере класс-файлов используемых конфигураций.

Конфигурация	Размер класс-файлов, байт
CLDC	529 780
MEEP	3 681 690
CLDC + Hello World	530 121
CLDC + EEMBC	872 164

Для каждой конфигурации был измерен размер образа при использовании перечисленных далее способов оптимизации.

- Оптимизация none — без инициализации классов и удаления методов, полей и классов.
- Оптимизация init — с инициализацией классов, без удаления методов, полей и классов. Для выбора инициализируемых классов использовался критерий, описанный в разд. 2.
- Оптимизация eliminate — без инициализации классов, с удалением недостижимых методов, полей и классов. Для анализа достижимости методов использовался алгоритм RTA.
- Оптимизация init + eliminate — с инициализацией классов и удалением недостижимых методов, полей и классов. Для анализа достижимости и инициализации классов использовался предложенный в данной статье алгоритм.

Результаты измерений представлены в табл. 1. Использование предложенного алгоритма в открытой модели сокращает размер образа на 3,3...11,8 %. В закрытой модели эффективность способов оптимизации сильно зависит от приложения. Для измеренных приложений уменьшение размера образа составляет 56,1...95,6 %.

В большей степени размер образа уменьшается за счет удаления недостижимых методов, полей и классов. Разница между оптимизациями eliminate и init + eliminate показывает влияние инцилизации классов на размер образа. Вклад инициализации классов в уменьшение размера незначителен. Этот факт объясняется тем, что лишь небольшая часть инициализированных классов содержит непустые

[2] JSR — Java Specification Request, спецификация стандарта Java-платформы. Упомянутые спецификации доступны на сайте Java Community Process: http://www.jcp.org

Таблица 3

Время инициализации системных классов конфигурации MEEP, мс

Платформа	Вид оптимизации	
	eliminate	init + eliminate
Raspberry Pi	15	8 (53,3)
Freescale FRDM-K64F	60	41 (68,3)

Примечание: в столбце init + eliminate в скобках указано время в процентах относительно значения в столбце eliminate.

статические инициализаторы. Информация о числе инициализированных классов приведена в табл. 2.

Также было измерено влияние инициализации классов на время инициализации на этапе исполнения. Для этого измерялось время инициализации системных классов конфигурации MEEP с использованием оптимизаций init и init + eliminate. Временем инициализации считалось время с начала исполнения Java-кода до начала исполнения кода приложения. Измерения проводили на платформах Raspberry Pi (ARM11 700 MHz, 256 MB RAM) и Freescale FRDM-K64F (ARM Cortex M4 120 MHz, 256 KB RAM). Результаты измерений представлены в табл. 3. Ранняя инициализация классов сокращает время инициализации на 31,6...46,6 % в зависимости от платформы.

8. Обзор смежных работ

Помимо алгоритмов CHA и RTA, описанных в разд. 1, существуют и другие алгоритмы построения замыкания графа вызовов. Для оценки возможных типов объектов в любой точке программы алгоритм RTA использует одно множество instantiable_classes. Более точные алгоритмы строят граф потока данных программы и для каждого узла в графе вычисляют множество достигающих типов. Алгоритмы отличаются точностью моделирования потока данных. Например, алгоритм XTA, описанный в работе [8], моделирует поток данных между методами программы. В языке Java объекты передаются между методами через аргументы, возвращаемые значения и ссылки в куче (ссылки могут содержаться в полях объектов и элементах массивов). Алгоритм не учитывает порядок выполнения программы и состояние локальных переменных.

Примером более точного алгоритма является алгоритм Variable-Type Analysis (VTA) [7], который моделирует поток данных между переменными. Алгоритм анализирует операции присваивания между локальными переменными, полями объектов и элементами массивов. Разные экземпляры одной переменной моделируются с помощью одного узла в графе. При анализе не учитывается порядок выполнения программы. Авторами также предложена вариация алгоритма под названием Declared-Type Analysis (DTA), которая моделирует разные переменные одного типа с помощью одного узла в графе.

В работе [5] рассмотрены алгоритмы, которые учитывают контекст исполнения при моделировании потока данных между переменными. При этом для каждой переменной в графе потока данных может существовать несколько узлов, соответствующих разным контекстам исполнения. К таким алгоритмам относятся k-Control Flow Analysis (k-CFA), Cartesian Product Algorithm (CPA), Simple Class Set (SCS).

Более точные алгоритмы незначительно уменьшают число достижимых методов, но существенно уменьшают число ребер в графе вызовов. Сокращение числа ребер в графе позволяет более точно идентифицировать мономорфные вызовы, которые впоследствии могут быть оптимизированы. Все описанные выше алгоритмы могут быть дополнены инициализацией классов.

Раздельная инициализация часто используется для оптимизации программ для встроенных систем. Например, в языке Virgil [9] инициализация программы осуществляется на этапе компиляции. Язык не поддерживает динамического создания объектов, все используемые программой объекты должны быть созданы во время инициализации. При инициализации исполняются конструкторы компонентов, которые могут содержать произвольный код. Инициализация осуществляется до анализа достижимости методов. Для анализа достижимости методов, полей и объектов используется алгоритм Reachable Member Analysis (RMA). Данный алгоритм расширяет алгоритм RTA анализом достижимости полей и объектов. С использованием этого алгоритма анализируются чтения полей и вычисляется множество объектов, достижимых для приложения, аналогично тому, как это делается в предложенном авторами алгоритме. По существу, предложенный алгоритм является развитием идей алгоритма RMA применительно к языку Java.

Другой пример применения алгоритма RMA для анализа Java-программ описан в работе [10]. Авторами описывается система ExoVM, автоматически специализирующая виртуальную машину для заданного приложения. Специализация осуществляется за счет анализа достижимости не только сущностей языка, но и отдельных компонентов виртуальной машины. В работе определяются отношения достижимости между методами, полями, объектами, классами и компонентами виртуальной машины. В итоговую виртуальную машину включаются только достижимые из точек входа приложения конструкции.

В отличие от предложенного в настоящей работе алгоритма, ExoVM инициализирует все классы до анализа достижимости методов. Такая инициализация может приводить к изменению видимого поведения программы за счет инициализации неиспользуемых классов и объектов. Также при удалении полей в ExoVM не учитываются побочные эффекты, связанные с потерей достижимости объектов.

Заключение

Предложен алгоритм анализа достижимости методов, осуществляющий выборочную инициализацию используемых классов. Данный алгоритм дополняет алгоритм RTA инициализацией классов. При этом инициализируются только те классы, ранняя инициализация которых не может изменить поведение программы. Для выбора таких классов предложена консервативная эвристика.

В отличие от алгоритма RTA, предложенный алгоритм учитывает созданные при инициализации классов объекты. Чтобы определить, какие из этих объектов будут достижимы для приложения, предложенный алгоритм анализирует чтения полей в достижимом коде.

В статье также предложены алгоритмы анализа удалимости полей и классов, которые учитывают существующие на момент анализа объекты. Если через ссылку в поле достижим объект, уничтожение которого приложение может отследить, поле нельзя удалять, даже если оно не используется в коде достижимых методов. В некоторых случаях алгоритм позволяет удалять поля, для которых есть операции записи, но нет операций чтения. Например, неиспользуемое поле, инициализированное в конструкторе или статическом инициализаторе, может быть удалено, если побочные эффекты записи в поле не влияют на видимое поведение программы.

Возможным развитием может быть использование глобального анализа потока данных. Такой анализ позволит уточнить анализ достижимости методов и удалимости полей за счет более точной оценки возможных типов значений времени исполнения.

Список литературы

1. **Пилипенко А. В., Плисс О. А.** Анализ достижимости методов при выборочной инициализации классов в программах на языке Java // Программная инженерия. 2014. № 8. С. 3—8.
2. **Aslam F., Fennell L., Schindelhauer C.** et. al. Optimized Java Binary and Virtual Machine for Tiny Motes // Distributed Computing in Sensor Systems (DCOSS). 2010. Vol. 6131. P. 15—30.
3. **Bacon D., Sweeney P.** Fast static analysis of C++ virtual function calls // ACM SIGPLAN Notices. 1996. Vol. 31, Issue 10. P. 324—341.
4. **Dean J., Grove D., Chambers C.** Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis // Proceedings of the 9th European Conference on Object-Oriented Programming (Aarhus, Denmark, August 7—11, 1995). Berlin, Heidelberg, New York, Springer, 1995. P. 77—101.
5. **Grove D., DeFouw G., Dean J., Chambers C.** Call Graph Construction in Object-Oriented Languages // ACM SIGPLAN Notices. 1997. Vol. 32, Issue 10. P. 108—124.
6. **Lindholm T., Yellin F.** The Java Virtual Machine Specification. SUN Microsystems. 1999.
7. **Sundaresan V., Hendren L., Razafimahefa C.** et al. Practical Virtual Method Call Resolution for Java // ACM SIGPLAN Notices. 2000. Vol. 35, Issue 10. P. 264—280.
8. **Tip F., Palsberg J.** Scalable Propagation-Based Call Graph Construction Algorithms // ACM SIGPLAN Notices. 2000. Vol. 35, Issue 10. P. 281—293.
9. **Titzer B.** Virgil: objects on the head of a pin // ACM SIGPLAN Notices. 2006. Vol. 41, Issue 10. P. 191—208.
10. **Titzer B., Auerbach J., Bacon D., Palsberg J.** The ExoVM System for Automatic VM and Application Reduction // ACM SIGPLAN Notices. 2007. Vol. 42, 6. June. P. 352—362.

Application Extraction and Selective Class Initialization for Java

A. V. Pilipenko, e-mail: artur.pilipenko@gmail.com, Saint Petersburg State University, 199034, Saint Petersburg, Russian Federation, **O. A. Pliss**, oleg.pliss@gmail.com, Oracle Corporation, CA 94065, Santa Clara, USA

Corresponding author:

Pilipenko Artur V., Postgraduate Student, Saint Petersburg State University, artur.pilipenko@gmail.com, Saint Petersburg, Russian Federation,
e-mail: artur.pilipenko@gmail.com

Received on May 10, 2016
Accepted on May 30, 2016

Application extraction, i.e. elimination of unused methods, fields and classes during Java program romization is considered. Eager class initialization is one of the romization optimizations. Class initialization in Java involves executing class initializers which can contain arbitrary Java code. Objects created during class initialization might affect the reachability of methods, fields and classes in non-trivial ways. Existing analysis algorithms don't take these objects into account.

We propose a reachability analysis algorithm which selectively initializes classes and takes objects created by class initializers into consideration. This algorithm is based on Rapid Type Analysis (RTA) algorithm which keeps track of classes which can be instantiated by reachable methods. The set of instantiable classes is used to determine which methods can be invoked by virtual and interface calls. The algorithm we propose also keeps track of classes which can be initialized by reachable methods. The algorithm initializes a subset of these classes using a simple heuristic to choose which classes are safe to initialize. All the objects which remain reachable after class initialization are considered to be reachable for the application. These objects could be used for virtual and interface calls. Therefore classes of reachable objects are included into the set of instantiable classes.

We also propose a field reachability analysis algorithm which takes live objects into account. Field removal might affect object reachability and might cause some objects to be collected by GC. In some cases that can be observable by the application even if it never accesses the object explicitly. For example, if the object is finalizable or a special reference is held for this object. The proposed algorithm takes these dependencies into account.

Finally we propose a class reachability analysis algorithm which takes aforementioned analyses into account.

Keywords: *Java, virtual machine, romization, class initialization, application extraction, reachability analysis, GC*

For citation:

Pilipenko A. V., Pliss O. A. Application Extraction and Selective Class Initialization for Java, *Programmnaya Ingeneria,* 2016, vol. 7, no. 8, pp. 339—350.

DOI: 10.17587/prin.7.339-350

References

1. **Pilipenko A., Pliss O.** Analiz dostizhimosti metodov pri vyborochnoj inicializacii klassov v programmah na jazyke Java (Method reachability analysis and selective class initialization for Java programs), *Programmnaya Ingeneria,* 2014, no. 8, pp. 3—8 (in Russian).

2. **Aslam F., Fennell L., Schindelhauer C.** et. al. Optimized Java Binary and Virtual Machine for Tiny Motes, *Distributed Computing in Sensor Systems (DCOSS)*, 2010, vol. 6131, pp. 15—30.

3. **Bacon D., Sweeney P.** Fast static analysis of C++ virtual function calls, *ACM SIGPLAN Notices*, 1996, vol. 31, no 10. pp. 324—341.

4. **Dean J., Grove D., Chambers C.** Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis, *Proceedings of the 9th European Conference on Object-Oriented Programming* (Aarhus, Denmark, August 7—11, 1995), Berlin, Heidelberg, New York, Springer, 1995, pp. 77—101.

5. **Grove D., DeFouw G., Dean J., Chambers C.** Call Graph Construction in Object-Oriented Languages, *ACM SIGPLAN Notices*, 1997, vol. 32, no. 10, pp. 108—124.

6. **Lindholm T., Yellin F.** *The Java Virtual Machine Specification*. SUN Microsystems, 1999.

7. **Sundaresan V., Hendren L., Razafimahefa C.** et al. Practical Virtual Method Call Resolution for Java, *ACM SIGPLAN Notices*, 2000, vol. 35, no. 10, pp. 264—280.

8. **Tip F., Palsberg J.** Scalable Propagation-Based Call Graph Construction Algorithms, *ACM SIGPLAN Notices*, 2000, vol. 35, no. 10, pp. 281—293.

9. **Titzer B.** Virgil: objects on the head of a pin, *ACM SIGPLAN Notices*, 2006, vol. 41, no. 10, pp. 191—208.

10. **Titzer B., Auerbach J., Bacon D., Palsberg J.** The ExoVM System for Automatic VM and Application Reduction, *ACM SIGPLAN Notices*, 2007, vol. 42, no. 6, pp. 352—362.

Приложение

```
// Анализ достижимости методов
function find _ reachable _ methods {
  reachable _ methods, new _ reachable _ methods = entry _ points;
  indirect _ inocations = ∅;
  instantiable _ classes = ∅;
  initializable _ classes = ∅;

  while new _ reachable _ methods != ∅ and
     new _ initializable _ classes != ∅ {
  while new _ reachable _ methods != ∅ {
     method = @new _ reachable _ methods; // @ - операция извлечения
                        // и удаления произвольного
                        // элемента из множества
     for each inst in method {
      switch inst {
       case direct _ call:
        add(reachable _ methods, new _ reachable _ methods,
          inst.callee);
       case indirect _ call:
        add(indirect _ invocations, new _ indirect _ invocations,
          inst.callee);
       case new:
        add(instantiable _ classes, new _ instantiable _ classes,
          inst.class);
       case class _ init:
        add(initializable _ classes, new _ initializable _ classes,
          inst.class);
```

```
        case field _ read:
         read _ fields U= {inst.field};
        case putstatic:
         putstatic _ fields U= {inst.field};
        case object _ field _ write:
         written _ object _ fields U= {inst.field};
        case ldc:
         add(instantiable _ classes, new _ instantiable _ classes,
           inst.class);
         add(initializable _ classes, new _ initializable _ classes,
           inst.class);
        case class _ refernce:
         referenced _ classes U= {inst.class};
      }
      for each exception _ class in method.exception _ table
        referenced _ classes U= {exception _ class};
     }
}

for each class in new _ initializable _ classes {
   if can _ initialize(class)
       class.initialize();
   else
       add(reachable _ methods, new _ reachable _ methods,
        class.clinit);
}

gc(); visit _ objects();

add _ matching _ methods(new _ indirect _ invocations,
             instantiable _ classes);
add _ matching _ methods(indirect _ invocations,
             new _ instantiable _ classes);

for each class in new _ instantaible _ classes {
   add(reachable _ methods, new _ reachable _ methods,
       class.finalize);
}

   new _ indirect _ invocations = ∅;
   new _ instantiable _ classes = ∅;
  }
}

function add(set, working _ set, item) {
   if item not in set
     set, working _ set U= {item};
}

function visit _ objects() {
   for each finalizable object
     add(instantiable _ classes, new _ instantiable _ classes,
       object.class);

reachable _ objects, new _ reachable _ objects = rootset;
while new _ reachable _ objects != ∅{
   object = @new _ reachable _ objects;
   add(instantiable _ classes, new _ instantiable _ classes,
```

```
      object.class);
if is _ special _ reference(object)
   add(reachable _ objects, new _ reachable _ objects,
     object.referent);
for each reference in object {
  if field(reference) in read _ fields
   add(reachable _ objects, new _ reachable _ objects,
     object.field);
  }
 }
}

function add _ matching _ methods(methods, classes) {
   for each indirect _ invocation in methods
     for each subtype of indirect _ inocation.holder
       if subtype in classes {
         method = find(indirect _ inocation, subtype);
         add(reachable _ methods, new _ reachable _ methods,
           method);
   }
}

// Анализ удалимости полей
function find _ unremovable _ fields {
   unremovable _ fields = read _ fields;
   for each field in putstatic _ fields
     if !is _ initialized(field.holder)
       unremovable _ fields ∪= {field};

if special _ reference _ classes ∩ instantiable _ classes != ∅{
   for each object in heap
     for each field in object
       if object.field is reference
         unremovable _ fields ∪= {field};
     unremovable _ fields ∪= written _ fields;
} else {
   initialize _ may _ refer _ to();
   initialize _ refer _ to();

   ref _ to _ finalizable _ classes, new _ ref _ to _ finalizable _ classes =
     finalizable _ classes ∩ instantiable _ classes;
   while new _ ref _ to _ finalizable _ classes != ∅{
     while new _ ref _ to _ finalizable _ classes != ∅{
       class = @new _ ref _ to _ finalizable _ classes;
       for each field in may _ refer _ to(class) ∩
         (refer _ to(class) ∪ written _ fields)
       add(ref _ to _ finalizable _ fields,
         new _ ref _ to _ finalizable _ fields, field);
}

while new _ ref _ to _ finalizable _ fields != ∅{
   field = @new _ ref _ to _ finalizable _ fields;
   if field is non-static
     for each subclass of field.holder
       if subclass in instantiable _ classes
         add(ref _ to _ finalizable _ classes,
           new _ ref _ to _ finalizable _ classes, subclass);
  }
}
```

```
      unremovable _ fields U= ref _ to _ finalizable _ fields;
    }
  }

    function initialize _ may _ refer _ to {
      for each class
        for each field in class
         for each subtype of field.type
          may _ refer _ to(subtype) U= {field};
  }

  function initialize _ refer _ to {
     for each object in heap
       for each field in object
         if object.field is reference {
           referent = object.field;
           refer _ to(referent.class) U= {field};
        }
  }

  // Анализ удалимости классов
  function find _ unremovable _ classes {
     unremovable _ classes = referenced _ classes U instantiable _ classes;
     for each field in unremovable _ fields
       unremovable _ classes U= {field.holder};
  }
```

ИНФОРМАЦИЯ

С 11 по 14 октября 2016 г. на базе д/о Ершово (Московская область) состоится XVIII Международная конференция "Аналитика и управление данными в областях с интенсивным использованием данных" (DAMDID/RCDL'2016). Конференция планируется как **мультидисциплинарный форум** исследователей и практиков из разнообразных областей науки, содействующий сотрудничеству и обмену идеями в сфере анализа и управления данными в условиях их интенсивного использования. Ожидается, что подходы к анализу данных и управлению данными, развиваемые в конкретных областях X-информатики (таких как X=астро, био, гео, нейро, медицина, физика, химия, и пр.), социальных наук, а также различных отраслей информатики, промышленности, новых технологий, финансов и бизнеса составят существенный вклад в контент конференции.

Конференция DAMDID была образована в 2015 г. в результате трансформации конференции RCDL ("Электронные библиотеки: перспективные методы и технологии, электронные коллекции") в целях создания форума, рассматривающего насущные проблемы анализа и управления данными в ходе исследований в различных областях с интенсивным использованием данных (**data intensive domains — DID**). При таком преобразовании была обеспечена преемственность трансформированной конференции по отношению к RCDL, а также сохранено RCDL-сообщество, сформировавшееся в течение 16 лет успешной работы RCDL.

Сайт конференции: http://damdid2016.frccsc.ru/

УДК 004.04, 004.4, 004.7 DOI: 10.17587/prin.7.351-359

И. В. Галов, аспирант, мл. науч. сотр., e-mail: galov@cs.karelia.ru,
Петрозаводский государственный университет

Применение шаблонов проектирования программных приложений для реализации косвенного взаимодействия агентов в интеллектуальном пространстве

Рассмотрена проблема организации взаимодействия агентов в интеллектуальном пространстве. В случае косвенного взаимодействия агенты выполняют построение информационных сервисов на основе совместного накопления и обработки информации в разделяемом информационном хранилище. Предложен набор шаблонов проектирования для реализации такого взаимодействия в программных приложениях для широкого круга предметных областей. Применимость предложенных шаблонов демонстрируется на примере ранее разработанных программных приложений.

Ключевые слова: *интеллектуальное пространство, архитектура М3, многоагентная система, взаимодействие агентов, проектирование программного обеспечения, шаблоны взаимодействия*

Введение

Интеллектуальное пространство (ИП) образует вычислительную сервисно-ориентированную среду, адаптируемую под нужды пользователя [1, 2]. Основной задачей такой среды являются построение и доставка пользователям информационных сервисов с привлечением множества доступных вычислительных участников и источников данных. Решение этой задачи требует способов организации взаимодействия разнородных участников среды для совместной обработки информации из множественных источников. В данном исследовании рассматриваются ИП, реализуемые на основе архитектуры М3 [3—5].

Вычислительный участник ИП реализуется как программный агент, автономно работающий на некотором вычислительном устройстве в целях накопления в ИП разделяемой информации, ее преобразования или извлечения на ее основе новых знаний. В архитектуре М3 такие агенты получили название процессоров знаний (*knowledge processor*, *KP*). Построение сервиса сводится к совместному получению агентами *KP* нужного фрагмента информации, доставляемого затем пользователю в удобном для последнего виде. Таким образом, проблема организации взаимодействия агентов является ключевой при разработке программного приложения в виде ИП. В силу разнообразия возможных вариантов взаимодействия агентов в ИП и с учетом необходимости вовлечения большого числа участников и источников данных требуется создание новых методов для упрощения и автоматизации разработки программных приложений [6, 7].

В данной работе рассмотрен один из подходов к разработке взаимодействия агентов в ИП — на основе шаблонов проектирования. Частным случаем таких шаблонов являются шаблоны взаимодействия, описывающие организацию взаимодействия между агентами в ИП. В работе предложен набор шаблонов взаимодействия, который является обобщением опыта разработки программных приложений в виде ИП в соответствии с архитектурой М3 [8—10]. Применение прикладным разработчиком подходящего шаблона взаимодействия при проектировании собственного программного приложения упрощает процесс разработки, так как используются уже разработанные решения. Предложенные шаблоны взаимодействия также могут быть использованы как основа для генерации заготовок программного кода, в котором реализуется часть логики приложения, отвечающая за взаимодействие.

Организация взаимодействия между агентами

В многоагентных системах выделяют два основных вида взаимодействия между агентами: прямое и косвенное [11]. При прямом взаимодействии агенты взаимодействуют непосредственно друг с другом, а при косвенном — через некоторого посредника. Архитектура М3 определяет косвенное взаимодействие агентов *KP* в ИП через разделяемое информационное содержимое. При этом агенты могут работать на разнообразных устройствах вычислительной среды, включая мобильные и встроенные устройства,

а также вычислительные устройства глобальной сети Интернет. Доступ к информационному содержимому для агентов организуется с помощью брокера семантической информации (*semantic information broker*, *SIB*).

Информационное содержимое представлено с помощью модели данных RDF и состоит из множества троек вида "субъект, предикат, объект". Брокер *SIB* предоставляет агентам *KP* определенный набор операций для работы с тройками в информационном содержимом: добавление (insert), удаление (remove), обновление (update), запрос (query), подписка (subscribe). Операция подписки позволяет агенту *KP* подписаться на определенную информацию и получать ее изменения.

Структуру информационного содержимого ИП (в первую очередь, его семантическую составляющую) принято описывать с помощью OWL-онтологий. Онтология позволяет представить предметную область приложения в терминах классов, свойств и индивидов (экземпляров класса). Информация, представленная OWL-онтологией, может быть автоматически преобразована во множество RDF-троек [12]. Далее описание информации, которой обмениваются агенты *KP*, будет выполняться в терминах онтологических индивидов и их свойств.

Для того чтобы в результате взаимодействия агентов был построен требуемый информационный сервис, необходимо при проектировании программного приложения определить последовательность действий агентов [5]. В частности, в работе [13] предложена задача координации одновременного доступа агентов к разделяемой информации и предложено использовать такие механизмы, как семафоры, агенты-координаторы и ряд других дополнительных сущностей. Эти механизмы поддерживают организацию взаимодействия, но не определяют конкретных вариантов организации взаимодействия агентов в ИП для совместного построения информационного сервиса. Несмотря на то что в работе [5] предложена концептуальная модель построения сервиса в ИП за счет взаимодействия агентов *KP*, нет общего метода проектирования взаимодействия агентов в ИП. При разработке заданного программного приложения прикладному разработчику приходится разрабатывать собственные варианты декомпозиции приложения на агентов *KP* и действия каждого агента для участия в требуемом взаимодействии.

Одним из вариантов автоматизированной разработки программных приложений является шаблонный подход [14, 15], когда шаблоны проектирования позволяют описать используемые варианты взаимодействия в виде некоторого параметризуемого решения. Далее предложен набор шаблонов взаимодействия агентов *KP* в ИП, где каждый шаблон представлен в виде диаграммы последовательности.

Шаблоны взаимодействия агентов

Шаблоны взаимодействия описывают ситуации обмена информацией между агентами *KP* в ИП. В таких ситуациях может быть задействовано два или более агентов *KP*. Взаимодействие между двумя агентами *KP* будем считать простым, если оно состоит из публикации/изменения агентом-отправителем определенных индивидов в информационном содержимом и получении этих индивидов агентом-получателем, т. е. в результате передается информация от агента-отправителя агенту-получателю. Агент-отправитель может добавить информацию (операция insert), удалить (remove) или обновить (последовательное удаление и добавление). Агент-получатель может получить необходимую информацию с помощью операции запроса (query) или подписки (постоянного запроса, subscribe).

На рис. 1 изображена публикация агентом *KP A* и получение агентом *KP B* информации по запросу в виде индивида онтологии (рис. 1, *а*) и подписке на класс (рис. 1, *б*). Запрос инициируется самим агентом-получателем разово, в определенный момент времени. Использование операции подписки ориентировано на постоянное проактивное получение изменений интересующей агента информации. В зависимости от подписываемой информации возможны два варианта:

а) подписка на класс из онтологии (оповещения подписки происходят только при создании/удалении индивидов заданного класса);

б) подписка на изменение всех или конкретных свойств определенного индивида.

Подписка на класс подразумевает, что при появлении нового индивида определенного класса подписчик будет оповещен о появлении такого индивида, при этом он не будет оповещен об изменении свойств этого индивида. Для таких изменений используется подписка на свойства определенного индивида (рис. 2). Таким образом, при публикации новой (или удалении существующей) информации с помощью операции insert (remove) целесообразно использование подписки на класс, а при отслеживании изменений уже существующего индивида с помощью

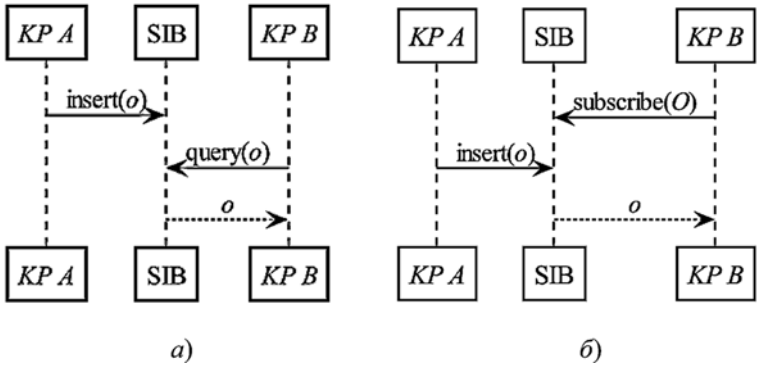


Рис. 1. Публикация и получение индивида *o*:
а — по запросу; *б* — по подписке на класс индивида *O*

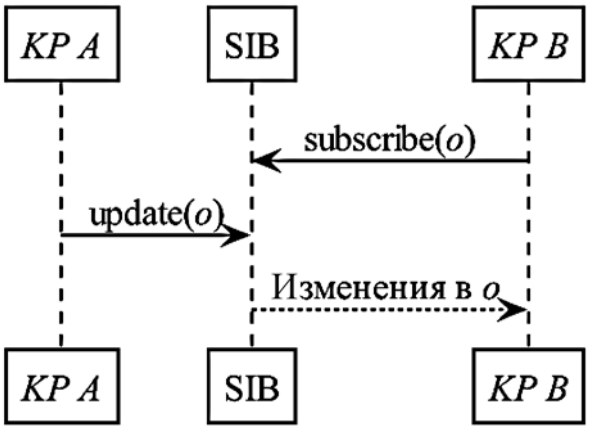


Рис. 2. Получение изменений индивида *o* по подписке

операции update — подписки на конкретные свойства индивида.

Косвенное взаимодействие допускает варианты, когда агенту-отправителю не известны агенты-получатели. Такое взаимодействие является частным случаем взаимодействия одного-ко-многим, когда агентами-получателями выступает множество одинаковых агентов $KP\ B_1$, ..., B_n (рис. 3). Такой подход чаще всего применяется в многопользовательских системах, где агенты-получатели являются клиентскими программами, работающими на устройствах конечного пользователя.

На основе простого взаимодействия можно определить шаблоны взаимодействия для трех агентов и более. Например, шаблон последовательного взаимодействия для трех агентов: *A* отправляет информацию *B*, *B* отправляет информацию *C*. Промежуточный агент *B* может выполнять одно из двух действий:

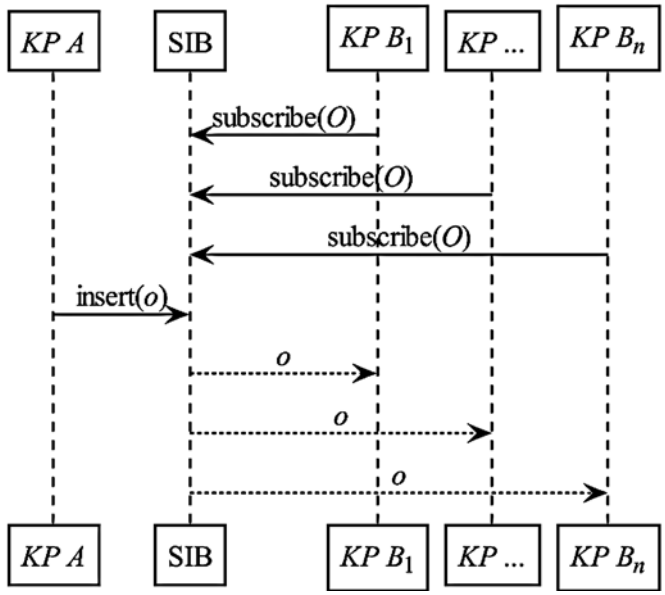
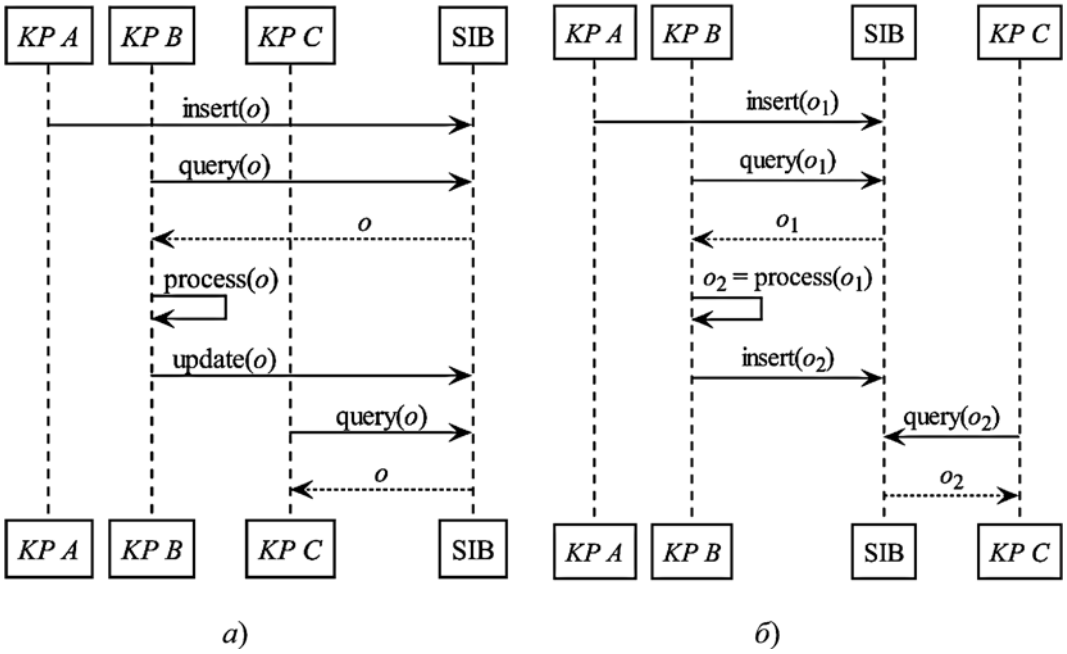


Рис. 3. Взаимодействие один-ко-многим

1) обновление существующего индивида *o* для последующей передачи агенту *C* (рис. 4, *а*),

2) публикацию нового индивида o_2 на основе полученного индивида o_1 (рис. 4, *б*).

Вариант 1 заключается в дополнительной обработке (process) опубликованного индивида *o* агентом *B* перед передачей агенту *C*. Такая обработка может заключаться в проверке/корректировке свойств индивида или добавлении новых свойств. В варианте 2



а) *б*)

Рис. 4. Последовательное взаимодействие

при обработке происходит создание нового индивида o_2 согласно другой онтологии на основе существующего индивида o_1. Агенты *B* и *C* могут получать индивидов как по запросу, так и по подписке. Данный подход может быть использован для преобразования информации (например, конвертирование в другой формат, логический вывод) в удобный для агента *C* вид (в соответствии с его онтологией). Следует отметить, что в таком случае агент *C* не располагает онтологией, описывающей информацию до преобразования.

Вариант публикации новой информации может быть расширен за счет выполнения передачи информации в обратную сторону. В таком случае агент *B* выступает в роли агента-посредника (медиатора) [16] (рис. 5). Агент-посредник осуществляет преобразование индивидов, тем самым организуя опосредованное взаимодействие между агентами *A* и *C*. Этот подход может применяться в случае, когда агенты *A* и *C* используют разные онтологии и агенту-отправителю не известна онтология агента-получателя. Агент *B*, выступающий в роли посредника, умеет работать с обеими онтологиями и осуществляет преобразование индивида o_1 из одной онтологии в индивида o_2 из другой онтологии и обратно.

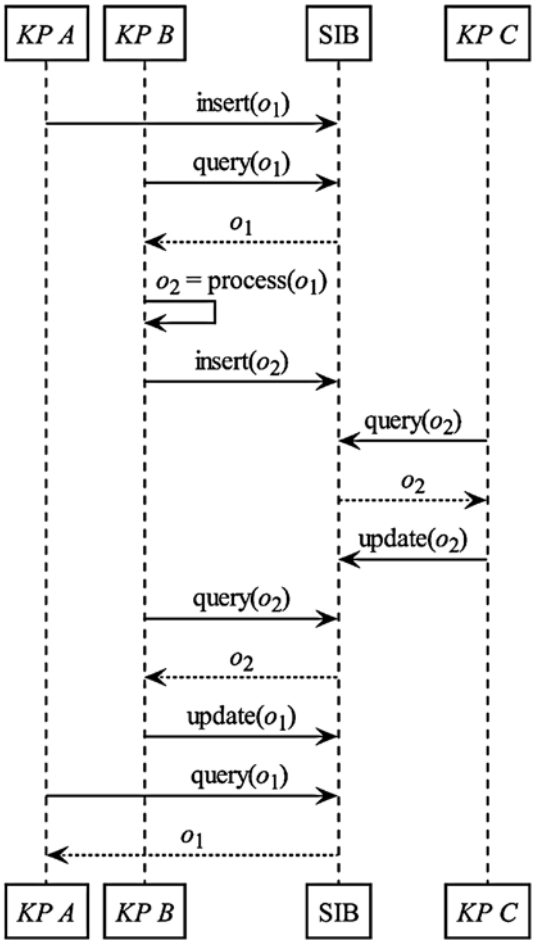


Рис. 5. Агент-посредник

Рассмотрим древовидные шаблоны взаимодействия, в которых происходит взаимодействие один-ко-многим и многие-к-одному [15]. На рис. 6 изображен шаблон взаимодействия "Фасад". При таком варианте взаимодействия агент *KP A* должен передать информацию нескольким агентам *KP*. Используется агент-посредник, который преобразует опубликованного агентом *A* индивида *o* во множество новых различных индивидов o_1, ... o_n, относящихся к соответствующему агенту-получателю (агенты C_1, ..., C_n). Данный вид взаимодействия отличается от простого взаимодействия один-ко-многим тем, что агенты-получатели являются разными агентами и могут использовать разные онтологии. Этот шаблон может применяться в ситуациях, когда необходимо преобразовать некоторую информацию и распространить среди множества функционально различных агентов.

Другим вариантом взаимодействия является взаимодействие многие-к-одному, которое представлено в шаблоне "Одиночка" (рис. 7). При таком взаимодействии агент *B* получает множество различных индивидов o_1, ... o_n от множества агентов-отправителей A_1, ..., A_n и преобразует их в одного индивида *o*, который передается агенту-получателю *C*. Агенты A_1, ..., A_n могут публиковать информацию, используя как одинаковую, так и разные онтологии. В таком случае агент *B* является концентратором и на основе полученной информации он создает новую информацию согласно онтологии агента *C*. Данный шаблон может применяться для сбора разнородной информации из различных источников, ее обработки и последующей доставки конечному пользователю.

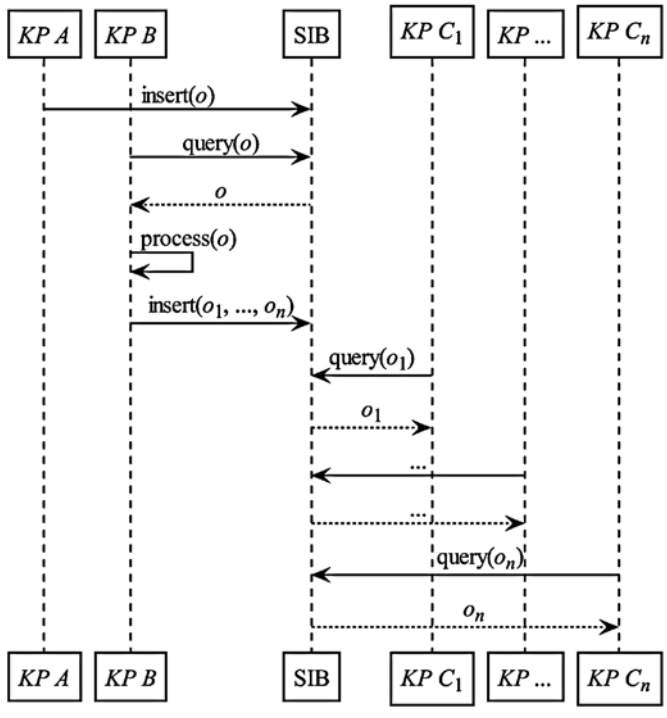


Рис. 6. Шаблон взаимодействия "Фасад"

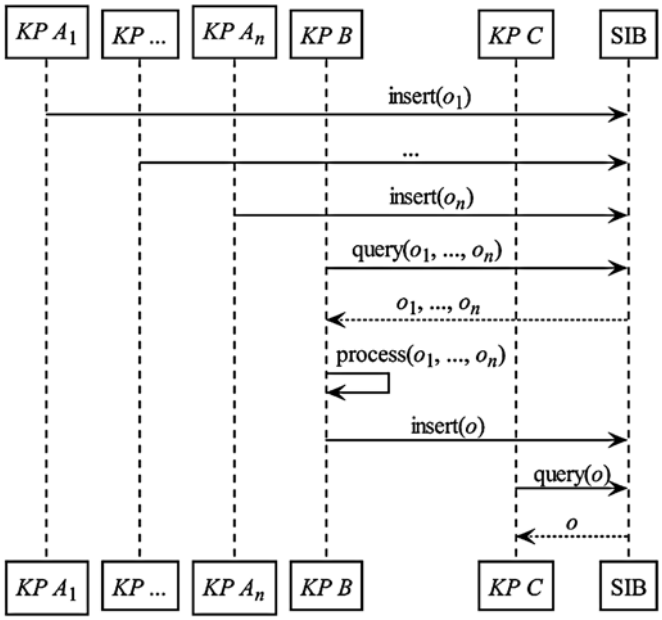


Рис. 7. Шаблон взаимодействия "Одиночка"

Следует отметить, что представленные шаблоны образуют иерархическую систему, где каждый последующий шаблон основывается на одном из предыдущих, как показано на рис. 8.

При проектировании программного приложения представленные шаблоны могут комбинироваться, образуя более сложные цепочки взаимодействия. В итоге в ИП формируются фрагменты важной для пользователя в данный момент информации, допускающие визуализацию пользовательским агентом *KP* для восприятия конечным пользователем.

Рис. 8. Дерево шаблонов

Примеры применения шаблонов

Шаблоны взаимодействия могут использоваться для автоматической генерации программного кода агентов *KP*, реализующих сервисы заданного программного приложения (рис. 9). При проектировании прикладной разработчик определяет между какими агентами какое взаимодействие требуется. Для каждого варианта взаимодействия выбирается подходящий шаблон. По шаблону взаимодействия создаются заготовки программного кода для всех агентов. Такая заготовка кода состоит из двух частей:

- реализация собственно взаимодействия;
- локальное состояние.

Реализация взаимодействия выполняется в терминах сетевых операций с информационным содержимым ИП. Локальное состояние определяется структурами данных для локального хранения агентом информации и возможными операциями по локальной обработке этой информации. Таким образом, прикладной разработчик концентрируется на реализации в агентах алгоритмов предметной области приложения, а не на реализации взаимодействия агентов в ИП.

Пример применения шаблона последовательного взаимодействия с созданием нового индивида предоставляется программным приложением M3-Weather [10]. Оно предназначено для отображения прогноза погоды относительно текущего местоположения пользователя. Выделено три вида агентов: клиентский агент, агент местоположения и агент погоды (рис. 10). Для получения прогноза погоды клиентский агент публикует текущие координаты устройства пользователя. На основе этих координат агент местоположения определяет ближайший населенный пункт. Затем агент погоды по названию населенного пункта запрашивает и публикует прогноз погоды, который затем доставляется клиентскому агенту. Для реализации таких взаимодействий между тремя агентами *KP* подходит шаблон последовательного взаимодействия с созданием нового индивида. На рис. 10 представлены шаблоны взаимодействия и кода для приложения M3-Weather. В качестве локального состояния агентов используются координаты, название города и прогноз погоды.

Пример применения шаблона последовательного взаимодействия с обновлением индивида предоставляется программным приложением туристического информационного сервиса планирования путешествий [17]. Данный сервис позволяет спланировать маршрут путешествия по заданным пользователем точкам маршрута. Выделено три агента *KP*: пользовательский агент, транспортный агент и агент для планирования расписания. На рис. 11 представлены шаблоны взаимодействия и кода агентов сервиса для получения расписания маршрута.

Пользователь с помощью пользовательского агента определяет целевые пункты путешествия (точки маршрута) и публикует эти данные в индивиде онтологического класса,

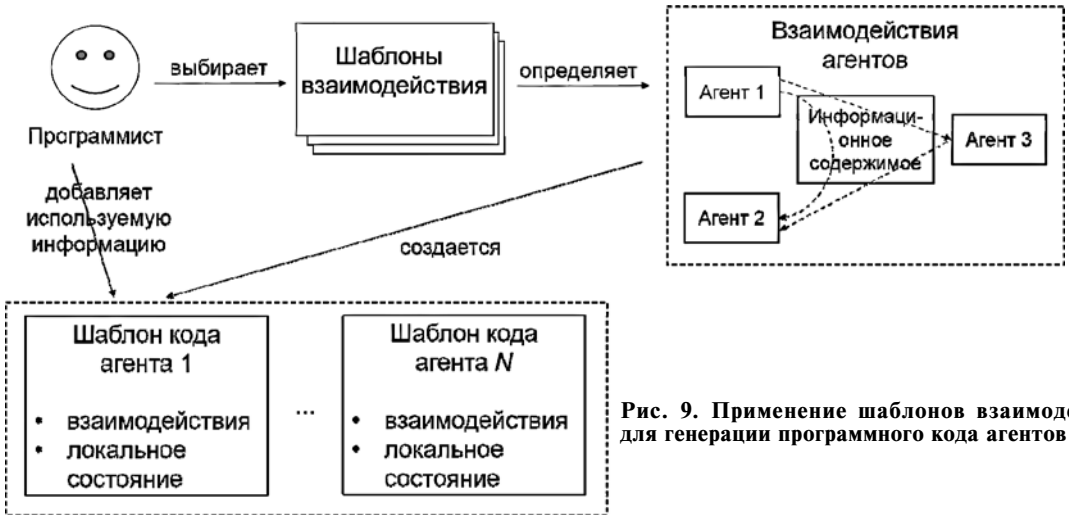


Рис. 9. Применение шаблонов взаимодействия для генерации программного кода агентов *KP*

описывающего маршрут. Транспортный агент формирует оптимальный маршрут обхода целевых пунктов путешествия и обновляет данные индивида класса "маршрут". Затем агент для планирования расписания на основании расписания транспорта рассчитывает время начала и окончания движения между точками маршрута. В результате, информационный сервис планирования поездки создает график, основанный на информации о маршруте, опубликованной пользователем.

Пример применения шаблона агента-посредника в интеграции интеллектуальных приложений предоставляется интеграцией программных приложений SmartScribo и SmartConference [9, 16]. Приложение SmartConference предназначено для автоматизированного проведения конференций и состоит из трех агентов: клиента, проектора и программы мероприятия. Агент, отвечающий за программу мероприятия, формирует порядок выступления докладчиков и стартует конференцию. Он взаимодействует с агентом-проектором, который показывает слайды текущего докладчика. Докладчик с помощью клиентского агента управляет переключением слайдов презентации.

Приложение SmartScribo используется для мобильного блоггинга, позволяя работать одновременно с несколькими блог-сервисами. Оно представлено тремя видами агентов *KP*: клиентом,

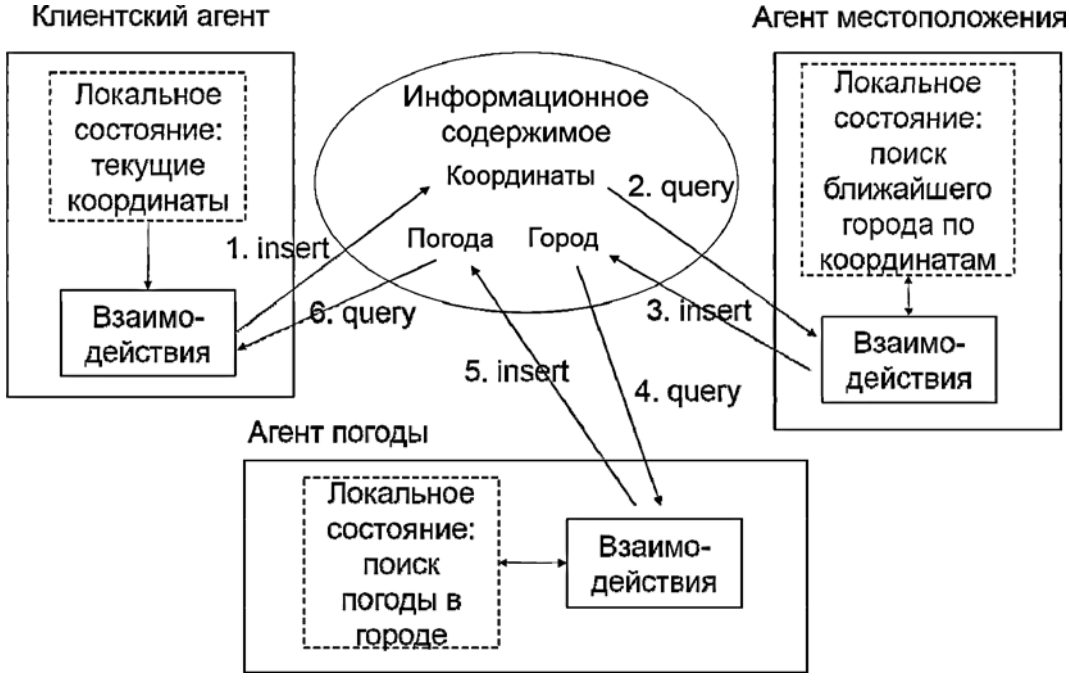


Рис. 10. Использование шаблона последовательного взаимодействия на примере приложения M3-Weather

Рис. 11. Использование шаблона последовательного взаимодействия на примере сервиса планирования путешествий

блог-процессором и блог-медиатором. Клиентский агент позволяет пользователю работать с записями в блогах. Блог-процессор осуществляет взаимодействие с определенным блог-сервисом: отправляет и получает записи из блогов пользователя. Блог-медиатор осуществляет дополнительную обработку блог-записей, например, ранжирование.

Агент-посредник на основе шаблона "посредник" интегрирует эти два приложения, расширяя возможности проведения конференции дискуссиями с использованием известных блог-сервисов социальных сетей. Для каждого выступления докладчика на конференции агент-посредник создает посты в блоге, в которых разворачивается дискуссия для данного выступления.

В приведенном примере агент-посредник преобразует информацию только в одном направлении. В то же время возможно расширение для выполнения взаимодействия и в обратном направлении: из SmartScribo в SmartConference. Например, комментарии к посту на блоге, посвященному текущему докладу, могут быть переданы из приложения SmartScribo в SmartConference для отображения на общем экране с расписанием проводимой конференции.

Заключение

Получено решение задачи построения и доставки информационных сервисов в ИП на основе применения шаблонов взаимодействия агентов. Предложен набор шаблонов, составляющих иерархическую систему. Эти шаблоны позволяют организовать цепочки взаимодействий между двумя и более агентами *KP*. На примерах ранее разработанных программных приложений демонстрируется применимость шаблонов взаимодействия для таких перспективных классов сервисно-ориентированных программных приложений, как туристические информационные сервисы и информационные сервисы поддержки проведения конференций. В целом, предложенные шаблоны взаимодействия позволяют упростить и частично автоматизировать разработку широкого круга программных приложений, реализуемых в виде ИП.

Работа выполнена при финансовой поддержке Минобрнауки России по заданию № 2014/154 на выполнение государственных работ в сфере научной деятельности в рамках базовой части государственного задания, НИР № 1481. Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 14-07-00252.

Список литературы

1. **Cook D. J., Das S. K**. How smart are our environments? An updated look at the state of the art // Pervasive and mobile computing. 2007. Vol. 3, N. 2. P. 53—73.

2. **Гаврилов А. В.** Искусственный домовой // Искусственный интеллект и принятие решений. 2012. № 2. С. 77—89.

3. **Honkola J., Laine H., Brown R., Tyrkko O.** Smart-M3 information sharing platform // Proc. IEEE Symp. Computers and Communications, Riccione, Italy, 22—25 June 2010. Washington: IEEE Computer Society, 2010. P. 1041—1046.

4. **Kiljander J., D'elia A., Morandi F.** et. al. Semantic interoperability architecture for pervasive computing and Internet of Things // IEEE Access. 2014. Vol. 2. P. 856—873.

5. **Корзун Д. Ж.** Формализм сервисов и архитектурные абстракции для программных приложений интеллектуальных пространств // Программная инженерия. 2015. № 2. С. 3—12.
6. **Ovaska E., Cinotti T., Toninelli A.** The design principles and practices of interoperable smart spaces // Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools. IGI Global, 2012. P. 18—47.
7. **Bartolini S., Milosevic B., D'Elia A.** et. al. Reconfigurable natural interaction in smart environments: approach and prototype implementation // Personal and Ubiquitous Computing. 2011. Vol. 16, N. 7. P. 943—956.
8. **Korzun D. G., Kashevnik A. M., Balandin S. I., Smirnov A. V.** The Smart-M3 Platform: Experience of Smart Space Application Development for Internet of Things // Proc. 15th International Conference on Internet of Things, Smart Spaces, and Next Generation Networks and Systems (NEW2AN 2015) and 8th Conference on Smart Spaces (ruSMART 2015), St. Petersburg, Russia, August 26—28, 2015. Springer International Publishing, 2015. P. 56—67.
9. **Korzun D. G., Galov I. V., Kashevnik A. M.** et. al. Integration of Smart-M3 applications: Blogging in smart conference // Proc. 11th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking (NEW2AN 2011) and 4th Conference on Smart Spaces (ruSMART 2011), St. Petersburg, Russia, August 22—25, 2011. Springer Berlin Heidelberg, 2011. P. 51—62.
10. **Samoryadova A., Galov I., Borovinskiy P.** et. al. M3-weather: A Smart-M3 world-weather application for mobile users // Proc. 8th Conf. of Open Innovations Framework Program FRUCT (FRUCT8). Lappeenranta, Finland, November 9—12, 2010. Saint-Petersburg: SUAI, 2010. P. 160—166.
11. **Городецкий В. И.** Самоорганизация и многоагентные системы. I. Модели многоагентной самоорганизации // Известия РАН. Теория и системы управления. 2012. № 2. С. 92—120.
12. **Корзун Д. Ж., Ломов А. А., Ванаг П. И.** Автоматизированная модельно-ориентированная разработка программных агентов для интеллектуальных пространств на платформе Smart-M3 // Программная инженерия. 2012. № 5. С. 6—14.
13. **Smirnov A., Kashevnik A., Shilov N.** et. al. Anonymous agent coordination in smart spaces: State-of-the-art // Proc. 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking (NEW2AN) and Second Conference on Smart Spaces (ruSMART 2009), St. Petersburg, Russia, September 15—18, 2009. Springer Berlin Heidelberg, 2009. P. 42—51.
14. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2007. 366 с.
15. **Billard E. A.** Pattern of agent interaction scenarios as use case maps // IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. 2004. Vol. 3, N. 4. P. 1933—1939.
16. **Korolev Y., Korzun D., Galov I.** Smart space applications integration: A mediation formalism and design for Smart-M3 // Proc. 12th International Conference on Internet of Things, Smart Spaces, and Next Generation Networking (NEW2AN 2012) and 5th Conference ruSMART 2012, St. Petersburg, Russia, August 27—29, 2012. Springer Berlin Heidelberg, 2012. P. 128—139.
17. **Kulakov K. A., Petrina O. B.** Ontological Model of Multi-Source Smart Space Content for Use in Cultural Heritage Trip Planning // Proc. 17th Conference of Open Innovations Association FRUCT (FRUCT17), Yaroslavl, Russia, April 20—24, 2015. IEEE, 2015. P. 96—103.

Application of Software Design Patterns for Implementation of Indirect Agent Interactions in a Smart Space

I. V. Galov, galov@cs.karelia.ru, Department of Computer Science, Petrozavodsk State University, Petrozavodsk, 185910, Russian Federation

Corresponding author:

Galov Ivan V., Postgraduate Student, Junior Researcher, Petrozavodsk State University, 185910, Petrozavodsk, Russian Federation,
e-mail: galov@cs.karelia.ru

Received on May 25, 2016
Accepted on May 31, 2016

The paper considers the problem of agent interaction arrangement in a smart space. Each smart space creates a computing service-oriented environment, which adapts for users' needs. The main purpose of such an environment is construction and delivery of information services to users. Service construction is performed by software agents interacting with each other and processing various information sources. Therefore, the problem of agent interaction arrangement is crucial for application development for smart spaces. Interactions among agents can be direct and indirect. In the case of indirect interaction, the agents construct information services based on cooperative information accumulation and processing in a shared information storage. Because of the diversity of different possible interactions and the necessity of involving a large number of agents it is necessary to elaborate new methods for simplifying and automating smart space application development. Recent research studies in this field consider only abstract conceptual models or solve narrow problems such as agent coordination for shared resources. There is no generic approach to design agent interaction in smart spaces. This paper considers an approach to arrange agent interactions based on software design patterns. We introduce a set of interaction patterns, which describe how to implement such interactions in software applications for a wide range of problem domains. The interaction patterns include patterns of information publishing and retrieving between two agents based on the query and subscription operations. Such patterns can be extended to one-to-many interactions where recipients are agents of the same type. More complicated patterns encompass sequential interaction among three or more agents where the transmitted information is processed and transformed by a mediator agent. The suggested patterns can be combined to achieve

non-trivial interaction chains. The applicability is shown using previously developed software applications for smart spaces. The proposed interaction patterns can also be used for code generation, hence allowing the developer to concentrate on agent processing algorithms implementation instead of interactions implementation.

Keywords: *smart spaces, M3 architecture, multi-agent systems, agent interaction, software design, interaction patterns*

Acknowledgements: *This research is financially supported by the Ministry of Education and Science of Russia within project no. 1481 of the basic part of state research assignment for 2014—2016. The reported study was supported by the Russian Foundation for Basic Research, research project no. 14-07-00252.*

For citation:

Galov I. V. Application of Software Design Patterns for Implementation of Indirect Agent Interactions in a Smart Space, *Programmnaya Ingeneria*, 2016, vol. 7, no. 8, pp. 351—359.

DOI: 10.17587/prin.7.351-359

References

1. **Cook D. J., Das S. K.** How smart are our environments? An updated look at the state of the art, *Pervasive and mobile computing*, 2007, vol. 3, no. 2, pp. 53—73.

2. **Gavrilov A. V.** Iskusstvennyj domovoj (Artificial domovoy), *Iskusstvennyj intellekt i prinyatie reshenij*, 2012, no. 2, pp. 77—89 (in Russian).

3. **Honkola J., Laine H., Brown R., Tyrkko O.** Smart-M3 information sharing platform, *Proc. IEEE Symp. Computers and Communications.* Riccione, Italy. Jun. 22—25, 2010, pp. 1041—1046.

4. **Kiljander J., D'elia A., Morandi F., Hyttinen P., Takalo-Mattila J., Ylisaukko-Oja A., Soininen J.-P., Cinotti T. S.** Semantic interoperability architecture for pervasive computing and Internet of Things, *IEEE Access*, 2014, vol. 2, pp. 856—873.

5. **Korzun D. G.** Formalizm servisov i arhitekturnye abstrakcii dlja programmnyh prilozhenij intellektual'nyh prostranstv (Service Formalism and Architectural Abstractions for Smart Space Applications), *Programmnaya Ingeneria*, 2015, no. 2, pp. 3—12 (in Russian).

6. **Ovaska E., Cinotti T., Toninelli A.** The design principles and practices of interoperable smart spaces, *Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools*, IGI Global, 2012, pp. 18—47.

7. **Bartolini S. Milosevic B., D'Elia A., Farella E., Benini L., Cinotti T. S.** Reconfigurable natural interaction in smart environments: approach and prototype implementation, *Personal and Ubiquitous Computing*, 2011, vol. 16, no. 7, pp. 943—956.

8. **Korzun D. G., Kashevnik A. M., Balandin S. I., Smirnov A. V.** The Smart-M3 Platform: Experience of Smart Space Application Development for Internet of Things, *Proc. 15th International Conference on Internet of Things, Smart Spaces, and Next Generation Networks and Systems (NEW2AN 2015) and 8th Conference on Smart Spaces (ruSMART 2015)*, Saint Petersburg, Russia, Aug. 26—28, 2015, pp. 56—67.

9. **Korzun D. G., Galov I. V., Kashevnik A. M., Shilov N. G., Krinkin K., Korolev Y.** Integration of Smart-M3 applications: Blogging in smart conference, *Proc. 11th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking (NEW2AN 2011) and 4th Conference on Smart Spaces (ruSMART 2011).* Saint Petersburg, Russia, Aug. 22—25, 2011, pp. 51—62.

10. **Samoryadova A., Galov I., Borovinskiy P., Kulakov K., Korzun D.** M3-weather: A Smart-M3 world-weather application for mobile users, *Proc. 8th Conf. of Open Innovations Framework Program FRUCT (FRUCT8).* Lappeenranta, Finland, Nov. 9—12, Saint Petersburg, SUAI, 2010, pp. 160—166.

11. **Gorodetskiy V. I.** Samoorganizatsiya i mnogoagentnye sistemy. I. Modeli mnogo-agentnoy samoorganizatsii (Self-organization and multi-agent systems. I. Multi-agent self-organization models), *Izvestiya RAN. Teoriya i sistemy upravleniya*, 2012, no. 2, pp. 92—120 (in Russian).

12. **Korzun D., Lomov A., Vanag P.** Avtomatizirovannaja model'no-orientirovannaja razrabotka programmnyh agentov dlja intellektual'nyh prostranstv na platforme Smart-M3 (Automated model-oriented development of software agents for smart spaces based on Smart-M3 platform), *Programmnaya Ingeneria*, 2012, no. 5, pp. 6—14 (in Russian).

13. **Smirnov A., Kashevnik A., Shilov N., Oliver I., Balandin S., Boldyrev S.** Anonymous agent coordination in smart spaces: State-of-the-art, *Proc. 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking (NEW2AN) and Second Conference on Smart Spaces (ruSMART 2009)*, Saint Petersburg, Russia, Sept. 15—18, 2009, pp. 42—51.

14. **Gamma E., Helm R., Johnson R., Vlissides J.** *Priemy ob'ektno-orientirovannogo proektirovaniya. Patterny proektirovaniya* (Techniques of object-oriented design. Design patterns), Saint Petersburg, Piter, 2007, 366 p. (in Russian).

15. **Billard E. A.** Pattern of agent interaction scenarios as use case maps, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 2004, vol. 34, no. 4, pp. 1933—1939.

16. **Korolev Y., Korzun D., Galov I.** Smart space applications integration: A mediation formalism and design for Smart-M3, *Proc. 12th International Conference on Internet of Things, Smart Spaces, and Next Generation Networking (NEW2AN 2012) and 5th Conference ruSMART 2012*, Saint Petersburg, Russia, Aug. 27—29, 2012, pp. 128—139.

17. **Kulakov K. A., Petrina O. B.** Ontological Model of Multi-Source Smart Space Content for Use in Cultural Heritage Trip Planning, *Proc. 17th Conference of Open Innovations Association FRUCT (FRUCT17)*, Yaroslavl, Russia. Apr. 20—24, 2015, pp. 96—103.

УДК 004.852 DOI: 10.17587/prin.7.360-372

В. А. Васенин, д-р физ.-мат. наук, проф., e-mail: vasenin@msu.ru,
В. А. Роганов, ст. науч. сотр, e-mail: var@msu.ru, НИИ Механики МГУ имени М. В. Ломоносова, г. Москва, **М. Д. Дзабраев,** студент, e-mail: dzabraew@gmail.com, МГУ имени М. В. Ломоносова

Методы автоматизированного анализа тональности текстов в средствах массовой информации

Рассмотрены подходы к решению задачи анализа текстов на естественном языке в части выявления их эмоциональной окраски по отношению к определенному субъекту той или иной сферы деятельности. Дан краткий анализ известных методов, которые могут быть использованы для решения этой задачи, описаны предложенный авторами подход и программные средства, положенные в основу разработанного ими макета информационного сервиса для анализа тональности публикаций в средствах массовой информации применительно к МГУ имени М. В. Ломоносова. Представлены и проанализированы результаты тестирования разработанного метода на представительной выборке публикаций.

Ключевые слова: *анализ текстов, тональность текста, анализ средств массовой информации, извлечение фактов, обучение с учителем, нейронные сверточные сети*

Введение

Средства массовой информации (СМИ) с использованием современных инфокоммуникационных технологий перманентно генерируют, аккумулируют и хранят огромные объемы сведений, имеющих отношение к тем или иным субъектам политической и экономической, научно-технической и хозяйственной, социальной и других сфер деятельности. Эти сведения оказывают влияние на формирование отношения отдельных людей, больших социальных групп к этим субъектам, влияют на их имидж в обществе. Как следствие, перечисленные факторы оказывают прямое или опосредованное воздействие на результаты их деятельности. С учетом отмеченных обстоятельств задача создания эффективных средств автоматизации процессов выявления текстов в СМИ, имеющих отношение к отдельным субъектам перечисленных выше сфер деятельности в обществе, приобретает особую актуальность.

В настоящей работе подходы к решению этой задачи представлены и анализируются применительно к МГУ имени М. В. Ломоносова (далее просто МГУ), который является большим и национально значимым субъектом научно-технической и образовательной деятельности не только в России, но и за ее пределами. Методы и средства ее решения предполагается использовать как составляющие отдельного сервиса, который представляется отдельным категориям пользователей информационно-аналитической системы (ИАС) "ИСТИНА" (Информационная Система Тематического Исследования НАукометрических данных [1]. Эта система разработана и развивается в рамках отдельного проекта в МГУ как одна из составляющих общей системы управления университетом. Основная задача, которая решается с помощью этой системы, — тематический анализ данных наукометрического содержания и смежных с ним в целях подготовки принятия управленческих решений.

В статье представлена постановка задачи, краткий анализ уже существующих подходов, которые могут быть использованы для ее решения. Предложены авторские методы и средства, положенные в основу разработанного для этого макета. Представлены и проанализированы результаты тестовых испытаний и перспективы дальнейших исследований на этом направлении.

Методы, при помощи которых можно решать задачу обработки текстов на естественных языках, можно разделить на три семейства. Первое семейство включает методы формального описания языка, построение грамматик, использование онтологий.

Второе семейство представляют алгоритмы обучения с учителем. В рамках этого семейства экспертом создается обучающая выборка, которая состоит из пар (объект, правильный ответ). Используя такую выборку, обучают классификатор, который будет получать на вход произвольные объекты и предсказывать для них ответы. В контексте работы, результаты которой представлены в настоящей статье, объектами являются тексты на естественном языке, а ответами будут метки классов "хороший", "плохой", "нейтральный".

Третье семейство — это методы обучения без учителя. Используя методы этого семейства, можно попытаться выявить связи между объектами без вмешательства эксперта. Затем, основываясь на связях

между объектами, можно разбить множество объектов на подмножества (кластеры).

В исследовании, результаты которого представлены далее, было решено использовать методологию обучения с учителем.

1. Тональность текста

Как было отмечено ранее, для использования методов обучения с учителем нужно создать обучающую выборку. Чтобы создать такую выборку, необходимо уточнить понятия классов "хороший", "плохой", "нейтральный".

Будем считать, что текст обладает:

- отрицательной тональностью, если он способствует ухудшению репутации МГУ;
- положительной тональностью, если он улучшает репутацию МГУ;
- нейтральной тональностью, если он ни способствует ухудшению, ни улучшает репутацию МГУ, несмотря на то что упоминание МГУ в нем присутствует.

С этих позиций тексту, имеющему отношение к МГУ, присваиваются следующие метки классов:

−1 = ухудшает репутацию МГУ;

+1 = улучшает репутацию МГУ;

0 = ни ухудшает, ни улучшает репутацию МГУ.

Зачастую при анализе текста возникают трудности с определением того факта, ухудшает ли текст репутацию или нет, улучшает ее или является нейтральным. Как следствие, два разных эксперта могут приписать одному и тому же тексту разные метки классов. В связи с этим обстоятельством предлагается уточнить, что представляет собой каждый из классов.

Будем относить статью к тому или иному классу, в зависимости от того, факт какого рода, имеющий отношение к МГУ, встретился в статье. Возможно, что статья будет вовсе не об МГУ, но в одном предложении может содержаться факт, который и будет задавать тональность по отношению к МГУ. После анализа трех сотен статей, опубликованных на сайте газеты "Московский комсомолец", такие факты удалось сгруппировать в несколько групп (табл. 1).

Может случиться, что статья не будет содержать ни один из фактов, представленных в табл. 1. Тогда эксперт должен самостоятельно выделить факт в тексте и отнести его к одному из классов.

Заметим, что возможны случаи, когда текст попадает в два класса. Например, рассмотрим вымышленный текст: "*В МГУ прошла конференция, посвященная машинному обучению, на которой подрались студенты*". Данный текст содержит как положительный факт проведения научной конференции, так и отрицательный, свидетельствующий о драке.

В данной работе на настоящем этапе исследований такие ситуации не рассматриваются. По этой причине, если классификатор получит на вход текст с двумя фактами из разных классов {+1, −1, 0}, то при

Таблица 1

Классы тональности

Тональность	Факты
Положительная	• Проведение общественных мероприятий, олимпиад • Упоминания сотрудников МГУ в СМИ в качестве экспертов • Выступления в МГУ "больших людей": президенты, министры, ученые • Ученые из МГУ что-то сделали в науке • МГУ или в МГУ что-то построят, например, музей • Сотрудники МГУ получают награды • МГУ опроверг что-то плохое о себе • Назначение выпускника МГУ на значимую должность
Отрицательная	• Самоубийство • Взятки • Криминал • Драки • Проблемы с полицией у сотрудников • Студенты сделали что-то, ухудшающее репутацию МГУ • Студент вступил в ИГИЛ • Пожары, взрывы на территории МГУ
Нейтральная	• Ограбили/избили сотрудника/студента вне МГУ • Пропажи людей вне МГУ • Происшествия около МГУ • Какие-либо события около территории МГУ, к которым МГУ не имеет отношения • Смерть сотрудников МГУ • Прочее

классификации заведомо будет допущена ошибка. Отметим, что проанализировав 300 статей из газеты "Московский комсомолец", в которых упоминался МГУ, не было встречено ни одной статьи, которая имела бы два факта из разных классов.

2. Постановка задачи

С функциональной точки зрения подлежащая решению на первом этапе задача состоит из перечисленных далее двух подзадач.

1. Создать классификатор, который в автоматическом режиме, без участия человека, будет классифицировать тональность (табл. 1) текстов.

2. Создать программные механизмы, позволяющие скачивать сообщения (статьи) из заранее определенных источников (веб-сайтов крупных СМИ), выбирать из них те, которые имеют отношение к МГУ, формируя на их основе коллекцию текстов — выборку на вход классификатора тональности.

3. Анализ подходов к решению задачи

Настоящий раздел посвящен краткому анализу методов, с помощью которых можно проводить классификацию текстов на естественном языке, а также — извлекать факты. Такой анализ необходим для более полного представления о результатах, полученных авторами для решения поставленной задачи и оценки перспектив дальнейших исследований на этом направлении.

Предобработка текста. Перед тем как осуществлять классификацию или извлечение фактов, текст можно подвергнуть предобработке. Два наиболее распространенных способа такой предобработки называют лемматизацией и стеммингом. Лемматизация представляет собой процесс, при котором каждое слово в тексте приводится к своей нормальной форме. Глаголы — к инфинитиву, существительные — к именительному падежу единственного числа и т. д. Задачу лемматизации можно решать с помощью инструментария mystem (https://tech.yandex.ru/mystem/). При операции стемминга от слова берется только его основа. Стемминг является более грубой операцией, при которой происходит бо́льшая потеря информации, чем при лемматизации.

Алгоритмы обучения с учителем. Все алгоритмы машинного обучения с учителем работают следующим образом. Предполагается, что существует зависимость $y = a(x)$, которая любому объекту x ставит в соответствие метку класса y. В контекте решаемой задачи объекты x — это сообщения из СМИ, а y является меткой тональности и принадлежит множеству $\{-1, +1, 0\}$. Зависимость $y = a(x)$ является неизвестной, и ее необходимо аппроксимировать, а именно — подобрать такую функцию $y = b(x)$, которая будет близка к $y = a(x)$ в некоторой метрике.

Для построения функции $y = b(x)$ применяют алгоритмы обучения с учителем. Для применения таких алгоритмов создается обучающая выборка, которая представляет собой набор пар (объект, метка класса). Применительно к решаемой задаче обучающая выборка создается экспертом (учителем) и будет состоять из пар (сообщение из СМИ; метка класса тональности).

Поскольку для создания функции $y = b(x)$ будет проводиться обучение ЭВМ, а ЭВМ работает с числами, то каждый объект x должен представляться числами. В качестве числового представления объекта x может быть использован вектор вещественных чисел. Числовое представление объекта x называется признаковым описанием объекта x. Пусть функция $y = f(x)$ будет давать признаковое описание объекта. Тогда искомая аппроксимация $y = b(x)$ запишется в виде $y = b(x) = c(f(x))$, где функция c принимает на вход вектор вещественных чисел и на выходе выдает метку класса. Следует заметить, что представление функции b в виде композиции функций c и f позволяет рассматривать задачу поиска функции b в виде двух подзадач. Первая подзадача отвечает за соотнесение объекту признакового описания, и вторая часть заключается в поиске функции $c(x)$.

Для нахождения функции $y = c(x)$ предполагается, что $c(x)$ принадлежит некоторому семейству функций $c(x, r)$, где r является векторным параметром. Таким образом, нужно подобрать параметр r так, чтобы $c(x)$ наилучшим образом (в рамках рассматриваемого семейства) аппроксимировала $a(x)$. О функции $a(x)$ известно только ее поведение на обучающей выборке. В связи с этим обстоятельством можно выбирать параметр r таким, чтобы $c(x, r)$ давала наименьшее число ошибок на обучающей выборке. В этом случае можно рассмотреть сумму $L(r) = [c(x_1, r) = y_1] + ... + [c(x_n, r) = y_n]$. Здесь x_i — это объекты обучающей выборки, а y_i соответствующая объекту x_i метка. Значение $[c(x_i, r) = y_i]$ равняется нулю в случае, если $c(x_i, r) = y_i$, и равно единице в противном случае. В идеальном случае нужно подобрать такой параметр r, чтобы $L(r)$ равнялась нулю. Это означает, что $c(x, r)$ не будет допускать ошибок на обучающей выборке. Однако обычно не удается найти такое r, чтобы ошибок на обучающей выборке не было вообще, поскольку такого r может не существовать. Поэтому процесс обучения заключается в том, что нужно найти такое r, которое обеспечит минимум функции $L(r)$. Функция $L(r)$ называется функционалом качества. Таким образом, процесс обучения сводится к минимизации функции $L(r)$.

Следует отметить, из того факта, что на обучающей выборке функция $c(x)$ может показывать хорошие результаты, не следует, что $c(x)$ будет показывать сколь-нибудь хорошие результаты на объектах, которые не попали в обучающую выборку.

Классические подходы. К классическим подходам относятся такие алгоритмы, как SVM, Логистическая регрессия, Решающие деревья [2]. Как было отмечено выше, алгоритмы машинного обучения требуют, чтобы каждому объекту ставился в соответствие вектор признаков (признаковое описание объекта). Упомянутые алгоритмы требуют, чтобы векторы признаков, которые ставят в соответствие текстам, имели одинаковую длину. Наиболее простым способом извлечения вектора признаков фиксированной длины является **мешок *n*-грамм**. Буквенной (словесной) n-граммой называется n последовательно идущих букв (слов).

Суть способа, основанного на мешке n-грамм, заключается в следующем. Из обучающей выборки выделяют все n-граммы (буквенные или словесные) заранее заданных длин. В результате этой операции получается словарь n-грамм. Например, пусть обучающая выборка состоит из одного текста *мама мыла раму*. Если выделить из него все буквенные 2-граммы, то получится следующий словарь:

мама мыла раму

↓

_м _р а_ ам ла ма му мы ра ыл

В качестве признакового описания текста (сообщения) берется вектор следующего вида. На i-м месте стоит 0 или 1. Причем 1 означает присутствие i-й n-граммы в тексте, а 0 — ее отсутствие. Развивая далее пример и используя имеющийся словарь 2-грамм, проиллюстрируем извлечение признакового описания текста *папа мыл раму*:

папа мыл раму

↓

_м _р а_ ам ла ма му мы ра ыл

1 1 1 1 0 0 1 1 1 1

Недостатком такого способа извлечения признаков является тот факт, что он не учитывает порядок следования n-грамм.

Другим распространенным подходом является **term frequency-inverse document frequency (tf-idf)** [3]. В этом подходе вектор признаков строится следующим образом. Пусть есть выборка документов (корпус) D. Корпус D состоит из документов $d_1, ..., d_m$. Сначала из слов корпуса D строится словарь A слов (словесных 1-грамм). В качестве A можно взять все слова из D. Обозначим элементы (слова) множества A через $t_1, ..., t_n$.

Затем документу d ставится в соответствие вектор длины $|A|$:

$$(\text{tfidf}(t_1, d, D), ..., \text{tfidf}(t_n, d, D)).$$

Здесь функция tfidf определяется соотношением $(\text{tfidf}(t_i, d, D) = \text{tf}(t_i, d) \cdot \text{idf}(t_i, D)$.

Функция $\text{tf}(t_i, d)$ определяется формулой

$$\text{tf}(t_i, d) = \frac{\left|\{t \in d | t = t_i\}\right|}{\left|\{t \in d\}\right|} =$$

$$= \frac{\text{Число вхождений } t_i \text{ в документ } d}{\text{Число слов в документе } d}.$$

Значение $\text{idf}(t_i, D)$ задается соотношением:

$$\text{idf}(t_i, D) = \log\left(\frac{|D|}{\left|\{d \in D | t_i \in d\}\right|}\right) =$$

$$= \frac{\text{Число всех документов в корпусе}}{\text{Число документов из } D, \text{ в которых встретилось слово } t_i}.$$

Смысл величины idf заключается в том, что если некоторое слово t будет употребляться почти во всех документах корпуса, то $\text{idf}(t, D)$ будет близка к нулю. Таким образом, будет понижаться вес общеупотребительных слов, которые употребляются вне зависимости от класса документа. Существуют различные вариации величины tfidf, например delta tfidf [4].

Существуют подходы [5], которые позволяют выделять из корпуса определения и словосочетания. Причем термины и словосочетания могут встречаться в тексте не в виде непрерывной последовательности слов, т. е. между частями словосочетания могут находиться другие слова. Например, из вымышленного предложения "*студентка из ведущего университета страны стала террористкой*" можно выделить термин "*студентка террористка*". Таким образом, можно рассматривать наличие или отсутствие того или иного термина или словосочетания в тексте в качестве признака.

Получив признаковое описание объекта, можно использовать любой из алгоритмов, предложенных в начале этого раздела. Более подробную информацию об алгоритмах классификации можно получить в работе [2].

Искусственные нейронные сети. В последнее десятилетие большое внимание ученых привлекли искусственные нейронные сети. Появились примеры их успешного применения в области обработки изображений и естественных языков.

Отметим, что искусственные нейронные сети являются алгоритмами обучения с учителем. Поэтому перед тем как будет описано устройство таких сетей, стоит остановиться на алгоритмах извлечения признаковых описаний (нефиксированной длины) из текстов.

Извлечение признакового описания из объекта можно делать как алгоритмами, которые не имеют отношения к обучаемой искусственной нейронной сети, так и с помощью самой нейронной сети. Извлечение признакового описания при помощи (обучаемой) нейронной сети заключается в том, что до начала обучения функция, которая ставит в соответствие объектам (в данном случае текстам) признаковое описание, неизвестна. И эта функция будет найдена в процессе обучения нейронной сети. Вместе с этим в результате обучения нейронная сеть будет способна предсказывать метки классов.

В настоящем обзоре не будем рассматривать методы извлечения признаков с помощью нейронной сети.

Важным отличием нейронных сетей от классических алгоритмов является то, что нейронные сети могут работать с признаковыми описаниями объектов нефиксированной длины. В связи с этим обстоятельством можно выделять признаковые описания следующим образом. Сначала определить функцию $f(t)$, которая поставит в соответствие слову естественного языка t вещественный вектор фиксированной длины. Затем, имея текст $T = t_1 ... t_n$, можно поставить ему в соответствие вектор $S = f(t_1) \oplus ... \oplus f(t_n)$, где $\oplus$ есть операция конкатенации, или матрицу $S = [f(t_1)^T, ..., f(t_n)^T]$. После этого S подается на вход нейронной сети для автоматической классификации.

Опишем методы, с помощью которых можно реализовать функцию f, позволяющую отобразить слова на векторы.

Наиболее простым методом является **1-hot sparse encoding** (https://en.wikipedia.org/wiki/One-hot). Согласно этому методу из обучающей выборки выделяются все слова, которые формируют словарь слов. Далее *i*-му слову из словаря ставится в соответствие вектор (0, ..., 0, 1, 0, ..., 0) с единицей на *i*-м месте. Функция $f(x)$, где x — слово, определяется следующим образом: если x совпадает с *j*-м словом из словаря, то на выходе $f(x)$ даст соответствующий *j*-му слову вектор; если x в словаре нет, то на выходе у функции $f(x)$ будет вектор из нулей. При этом следует отметить, что для применения такого кодирования нужно иметь достаточно большую обучающую выборку, чтобы при работе классификатора не получить много нулевых векторов.

Существует другой способ соотнесения слову вектора — с помощью программы **word2vec** (https://code.google.com/archive/p/word2vec/). Принцип работы этой программы заключается в том, что ей на вход подают большие массивы текстовых данных, которые могут не иметь отношения к обучающей выборке. На вход программы word2vec также подают размерность пространства, в которое будут отображаться слова. В результате работы этой программы каждому слову из массива текстов будет поставлен в соответствие вектор заданной длины. Причем близким по смыслу словам будут поставлены в соответствие близкие векторы. Отметим, что в процессе обучения программа word2vec самостоятельно определяет смысловую близость слов, без участия человека.

Большим преимуществом того, что близкие по смыслу слова отображаются в близкие точки линейного пространства состоит в следующем. Если в обучающей выборке не будет некоторого слова w, однако будет близкое по смыслу w', тогда возможна ситуация, при которой в процессе классификации алгоритм встретит w, и он сможет связать его с w'. Таким образом, в качестве функции $f(x)$ можно взять вектор, который выдаст word2vec на слове x.

Перед тем как рассматривать примеры применения нейронных сетей для обработки текстов, кратко напомним, как они устроены. Базовой составляющей нейронной сети является нейрон (рис. 1). Нейрон имеет несколько входов. На входы поступают числа, на рис. 1 эти числа обозначены как $\{x_i\}$. Далее число, поступившее на *i*-й вход, умножается на коэффициент m_i, и так для каждого *i*. Затем взвешенные входные сигналы суммируются и к ним применяется некоторая функция g, после чего число $g(\Sigma m_i x_i)$ подается на выход нейрона.

Нейронные сети состоят из слоев. Каждый слой состоит из нейронов. Самому первому слою подаются на вход входные данные $\{x_i\}$. Выход первого слоя представляет входные данные для второго слоя и т. д. (рис. 2).

Далее кратко остановимся на примере успешного применения нейронной сети для обработки текста. Нейронная сеть, представленная в работе [6], состоит из слоев нескольких типов. К их числу относятся: сверточный слой; dynamic k-max pooling-слой; слой, на котором применяется нелинейная функция; слой с операцией folding. Эти четыре типа слоев применяются последовательно (рис. 3; рис. 4, см. вторую сторону обложки).

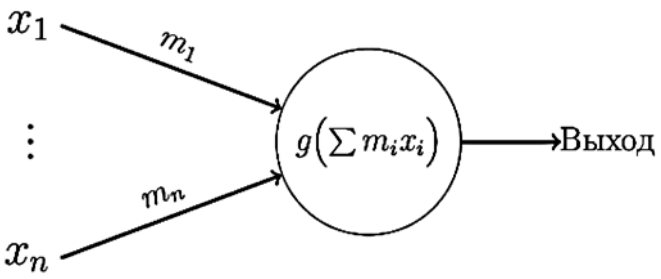


Рис. 1. Нейрон

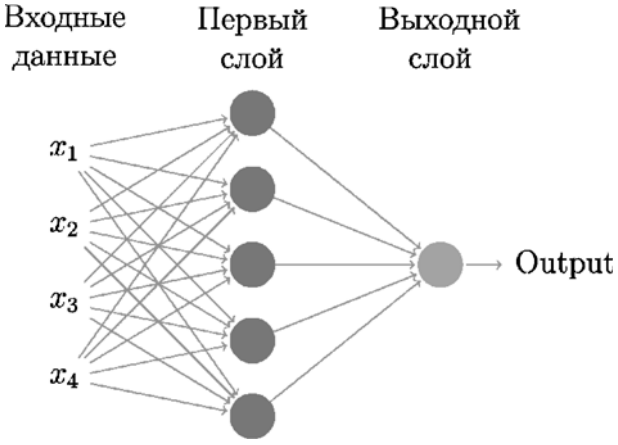


Рис. 2. Простейшая нейронная сеть

Рис. 3. Стопка слоев нейронной сети

На каждом из слоев реализуются перечисленные далее функции.

Сверточный слой. Пусть задано отображение $f(x)$ на словах, которое каждому слову ставит в соответствие вектор длины d. Пусть в тексте содержится n слов. Тогда запишем признаковое описание сообщения (текста) в виде матрицы

$$S = \begin{bmatrix} | & \dots & | \\ w_1 & \dots & w_n \\ | & \dots & | \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ \vdots & & \vdots \\ w_{d1} & \dots & w_{dn} \end{bmatrix}, \quad (1)$$

где w_i представляет собой вектор, поставленный в соответствие *i*-му слову.

Рассмотрим p-ю строчку матрицы S:

$$w_p = (w_{p1}, \dots, w_{pn})$$

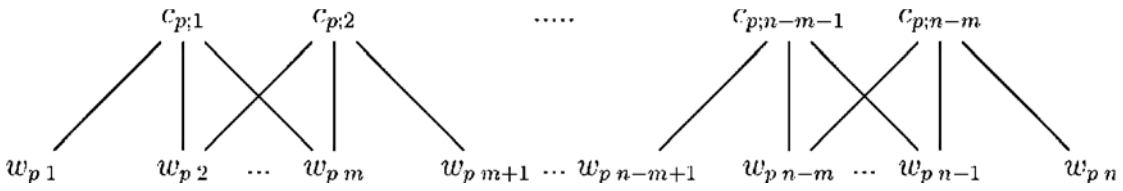


Рис. 5. Свертка узкого типа

и определим на ней операцию свертки. Свертка бывает двух типов: узкая и широкая. Иллюстрация свертки узкого и широкого типа представлена на рис. 5 и 6.

В алгебраической записи результат свертки выглядит следующим образом:

$$c_{p;e} = \sum_{i=1}^{m} m_{p;i} w_{pi+e}, \quad (2)$$

где m — ширина фильтра; m_i — вес фильтра.

Отметим, что набор коэффициентов $\{m_i\}$ не зависит от индекса e при вычислении $c_{p;e}$. Иными словами, набор $\{m_i\}$ один на всю строку w_p матрицы S. При узкой свертке индекс e пробегает диапазон от 1 до $(n - m)$. Таким образом, каждое число c_e зависит от m входных чисел. Недостаток свертки узкого типа заключается в том, что крайние числа w_{p1} и w_{pn} войдут в коэффициенты $\{c_{p;i}\}$ только по одному разу, при этом w_{p1} войдет в $c_{p;1}$, а w_{pn} войдет только в $c_{p;n-m}$. Вторые по счету от краев w_{p2} и w_{pn-1} войдут в $\{c_{p;i}\}$ по два раза. Коэффициенты w_{pi} из середины рассматриваемого вектора будут учитываться m раз при вычислении множества $\{c_i\}$. Таким образом, крайние коэффициенты вносят меньший вклад. Однако в предложениях естественного языка не наблюдается того факта, что слова по краям предложений менее важны, чем слова внутри предложения.

Для устранения отмеченного недостатка можно использовать свертку широкого типа. Для ее применения нужно расширить (увеличить размерность) вектора w_p, дописав $m - 1$ нулей слева и столько же нулей справа: $\hat{w}_p = (0, ..., 0, w_{p1}, ..., w_{pn}, 0, ..., 0)$. Затем к $\hat{w}_p$ применяется формула (2).

Выше была определена свертка, которая именуется одномерной. Теперь дадим определение многомерной свертки, которое выглядит следующим образом:

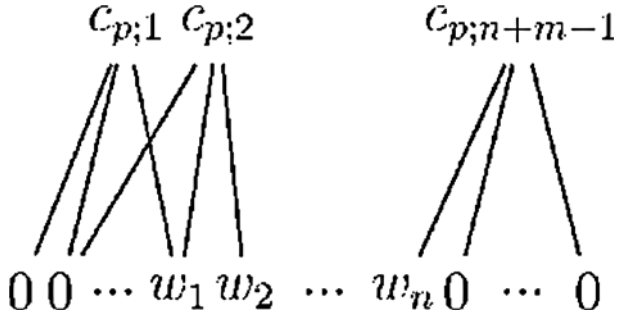


Рис. 6. Свертка широкого типа

$$c_{p;e} = \sum_{k=1}^{d} \sum_{i=1}^{m} m_{p;i}^{k} w_{ki+e}. \quad (3)$$

Таким образом, для вычисления $c_{p;e}$ в случае одномерной свертки окно ширины m двигалось только по строке с номером p. В случае многомерной свертки окно шириной m движется по всем строкам. Далее будем рассматривать только случай свертки широкого типа. После операции многомерной свертки входная матрица S будет преобразована в матрицу $C = (c_{p;e})$. При этом матрица C будет иметь больший размер, чем S.

Теперь отметим то обстоятельство, что операция свертки на каждом слое делается со своими весами. Дополним формулу (3) еще одним индексом, который отвечает за номер слоя:

$$c_{p;e}^{l} \sum_{k=1}^{d} \sum_{i=1}^{m} m_{p;i}^{k;l} w_{ki+e}^{l}.$$

Числа w_{ki}^{0} совпадают с матрицей S, т. е. с входными данными, а числа w_{ki}^{l} являются выходными данными с последнего слоя из набора слоев (см. рис. 3).

Таким образом сверточный слой определяется полностью.

Слой dynamic k-max pooling (далее DKMP). Данный слой принимает на вход результат сверточного слоя, т. е. матрицу C. Из каждой строки матрицы C берутся k максимальных элементов с сохранением порядка. Это означает, что часть элементов матрицы будут удалены, а элементы, которые останутся, сохранят свой порядок относительно друг друга:

$$\begin{bmatrix} c_{1;1} & \dots & c_{1;t} \\ \vdots & & \vdots \\ c_{d;1} & \dots & c_{d;t} \end{bmatrix} \rightarrow \begin{bmatrix} u_{11} & \dots & u_{1k} \\ \vdots & & \vdots \\ u_{d1} & \dots & u_{dk} \end{bmatrix}.$$

Число k при этом задается размером входных данных. Авторы данного метода предлагают брать $k = \max\left(k_{\text{top}}, \left\lceil \frac{L-l}{L} n \right\rceil\right)$, где l — номер текущего слоя с типом DKMP; L — общее число слоев типа DKMP; n — длина текста; а k_{top} — параметр, который ограничивает снизу выход слоя DKMP.

Слой с нелинейной функцией. Слой данного типа принимает на вход матрицу, полученную на выходе предыдущего слоя. Сначала к каждому элементу полученной матрицы прибавляется число b, а затем

к каждому элементу применяется нелинейная функция g:

$$\begin{bmatrix} u_{11} & \dots & u_{1k} \\ \vdots & & \vdots \\ u_{d1} & \dots & u_{dk} \end{bmatrix} \to \begin{bmatrix} g(u_{11}+b) & \dots & g(u_{1k}+b) \\ \vdots & & \vdots \\ g(u_{d1}+b) & \dots & g(u_{dk}+b) \end{bmatrix} = \begin{bmatrix} v_{11} & \dots & v_{1k} \\ \vdots & & \vdots \\ v_{d1} & \dots & v_{dk} \end{bmatrix}.$$

Folding. Далее авторы предлагают операцию folding. Суть этой операции заключается в том, что на вход она получает матрицу и складывает рядом стоящие строки. Результатом применения данной операции окажется матрица высотой $d/2$, а ширина останется как у матрицы, которая подавалась на вход:

$$\begin{bmatrix} v_{11} & \dots & v_{1k} \\ \vdots & & \vdots \\ v_{d1} & \dots & v_{dk} \end{bmatrix} \to \begin{bmatrix} v_{11}+v_{21} & \dots & v_{1k}+v_{2k} \\ v_{31}+v_{41} & \dots & v_{3k}+v_{4k} \\ \vdots & & \vdots \end{bmatrix}.$$

Слой с операцией folding полностью описан.

На последнем слое нейронной сети останется заранее заданное число выходов. Эти выходы подключаются к softmax-функции. Функция softmax представляется как

$$\sigma(z) = \left(\frac{e^{z_1}}{\sum_{k=1}^{n} e^{z_k}}, \dots, \frac{e^{z_n}}{\sum_{k=1}^{n} e^{z_k}} \right),$$

где i-я компонента вектора $\sigma(z)$ трактуется как оценка принадлежности поданного на вход нейронной сети объекта к i-му классу.

На рис. 4 (см. вторую сторону обложки) приведено графическое представление описываемой сверточной нейронной сети. Нижний слой на рис. 4 соответствует входным данным. Слой wide convolution обозначает свертку широкого типа. Fully connected layer означает применение функции softmax.

Таким образом, описание примера нейронной сети в его конспективном представлении можно считать завершенным. Вопросы применения этой сети более детально описаны в работе [6].

В качестве второго примера можно отметить сверточную нейронную сеть из работы [7]. Структура этой сети (рис. 7, см. вторую сторону обложки) схожа с описанной ранее (рис. 4, см. вторую сторону обложки).

Принципиальное отличие сети на рис. 7 от сети на рис. 4 заключается в использовании нескольких каналов (channels). Каналом называется матрица, которая получается в результате операции соотнесения числового описания тексту. В предыдущем примере каналом была матрица S (формула (1)). Авторы работы [7] попробовали использовать два разных канала. Первый получали, используя word2vec, а второй канал получался в процессе обучения нейронной сети. Применяя этот подход, авторы делали попытку предотвратить переобучение нейронной сети.

Более подробно результаты экспериментов с использованием этой нейронной сети представлены в работе [7].

Грамматики. Кроме классификации текстов не менее важным для решения поставленной выше основной задачи является исследование и реализация механизмов автоматизированного извлечения фактов из текста на естественном языке. Например, сотрудников МГУ цитируют в СМИ в качестве экспертов. Можно пытаться выделять факты, представленные в табл. 1, и на основе того или иного факта делать их классификацию.

Ниже приведены четыре отрывка из СМИ, в которых сотрудники МГУ упоминаются в качестве экспертов. Можно увидеть, что статьи подобного рода строятся по некоторым правилам. Сначала идет некоторое действие, предшествующее цитате, которое совершил сотрудник: "Как сообщил", "Как заявил" и т. д. Далее авторы публикации последовательно пишут должность сотрудника и затем — его имя.

- Цифра дня [**Как сообщил "МК"**] [**ученый МГУ им. Ломоносова, эксперт ООН по химической безопасности**] [**Валерий Петросян**] главным показателем достаточного количества выпитой влаги может служить цвет мочи. Она должна быть прозрачной. В противном случае каждый москвич должен потреблять на 1,5...2 литра больше ранее рекомендованных норм. То есть по 3...3,5 литра.
- МГУ: Нью-Йорк и Петербург смоет гигантской волной. Апокалипсические прогнозы обнародовали ученые из Московского государственного университета. [**Как рассказал**] [**научный сотрудник музея землеведения МГУ**] [**Николай Жарвин**], катастрофа придет в мир опять из Исландии. Вслед за вулканом из Исландии последует масштабное наводнение... наступления конца света надо не сейчас, а через несколько десятилетий, в 2030—2070 гг.
- Ученый: Тунгусская катастрофа — это извержение вулкана, Тунгусская катастрофа, произошедшая в 1908 году в Восточной Сибири, стала результатом не падения космического тела, а взрыва водорода и метана, выделившихся из жерла древнего вулкана. [**Об этом сказал**] [**доцент кафедры газодинамики мехмата МГУ**] [**Владимир Натяганов**]. Натяганов и его коллеги в статье, принятой к печати в журнале "Доклады Академии наук", ... катастрофы был крайне маловероятен.
- Дожди в Москве могут быть опасны для человека Кратковременные дожди, которые проходят в Москве после длительных периодов сухой и жаркой погоды, могут быть опасны из-за высокой кислотности и большого содержания вредных веществ, [**заявил**] [**научный сотрудник кафедры метеорологии и климатологии географического факультета МГУ, руководитель метеорологического отдела обсерватории МГУ**] [**Павел Константинов**]. "Из-за малого коли-

чества осадков воздушный бассейн столицы очень плохо очищается, и каждые выпадающие осадки являются... очень неблагоприятно сказывается на состоянии человеческого организма", — предупредил ученый.

Можно попытаться составить такие правила (грамматики) и проверять, удовлетворяет ли отрывок текста грамматике или нет. В соответствии со сделанным наблюдением о структуре текстов с цитатами, можно пытаться применить грамматики, представленные на рис. 8.

```
S       -> <ACTION> <ДОЛЖНОСТЬ> <ИМЯ>;
ACTION1 -> "заявить" | "рассказать" ;
ACTION  -> <частица> ACTION1;
ACTION  -> <предлог> <частица> ACTION1;
  ...
```

Рис. 8. Грамматики для распознавания цитирования сотрудников

Для работы с такого рода грамматиками существует инструментальное средство TomitaParser (https://tech.yandex.ru/tomita/). Оно способно обрабатывать контекстно-свободные грамматики. В качестве терминалов в грамматиках TomitaParser выступают части речи[1] и определенные пользователем терминалы. При написании грамматики можно потребовать, чтобы при условии успешного наложения грамматики на текст извлекались факты.

Известны успешные реализации данного подхода. Компания Yandex, используя TomitaParser в сервисе yandex-почта, извлекает в автоматическом режиме события из писем и предлагает добавить их в персональный календарь. Например, из предложения "*9 мая в 9:00 начнется парад Победы*" yandex-почта смогла выделить событие "парад Победы" и дату "9 мая 2016 г.".

4. Предлагаемый подход

Подход, описанный в настоящем разделе, был рассмотрен в работе [8], за исключением того, что проводилась классификация сообщений не на два, а на три класса. Для решения поставленной задачи использовалась следующая комбинация методов: для извлечения признакового описания сообщения (текста) — метод "мешок *n*-грамм"; для классификации сообщений — метод опорных векторов.

Был осуществлен перебор некоторого множества параметров, которые представлены в табл. 2. По результатам такого перебора были выбраны те параметры, которые показывали наилучшее качество.

В табл. 2 приведены параметры, которые влияли на процедуру извлечения признаков из текста. На рис. 9 приведены параметры, которые отвечали за настройку метода опорных векторов и обобщение

crammer-singer	→	default			
one-vs-rest	→	hinge	→	l_2	→ dual–T
		sqhi	→	l_1	→ dual=F
			→	l_2	→ dual–T,F

Рис. 9. Перебор параметров в методе опорных векторов

бинарной классификации на многоклассовую классификацию.

В первой строке табл. 2 приведены диапазоны *n*-грамм, которые перебирались. Извлечению подвергались все *n*-граммы из диапазонов (2,2), (2,3), ..., (2,12), ..., (11,12), (12,12). Были также опробованы как буквенные *n*-граммы (char), так и словесные (word) *n*-граммы. Параметр dfmin определяет, если *n*-грамма встречалась меньше, чем dfmin раз, то данная *n*-грамма не добавляется в словарь и, как следствие, не рассматривается как признак. Строка "тип признака" означает, что в качестве признака рассматривалось как наличие/отсутствие *n*-граммы, так и проявление *n*-граммы с учетом кратности. Все признаки были либо бинарные, либо количественные. Последняя строка в табл. 2 указывает, была выполнена лемматизация или нет.

Метод опорных векторов позволяет варьировать ряд параметров. Первый из них —обобщение бинарного классификатора на произвольное число классов с помощью алгоритмов, стратегии которых их авторы обозначают аббревиатурами ovr (one-vs-rest — "один против всех"), ovo (one-vs-one — "один против одного") и cr (crammer-signer) [9, 10]. Кратко поясним, что представляют собой стратегии one-vs-rest и one-vs-one (стратегия crammer-singer [9] сложна и ее описание выходит за рамки настоящей работы).

Стратегия ovr применяется для классификаторов, которые выдают оценку принадлежности объекта к классам. Это означает, что классификатор не просто выдает метку класса, а выдает несколько чисел — оценки принадлежность объекта к каждому классу. Если возникает необходимость обобщить бинарный классификатор на классификацию *k* классов, то для этого обучают *k* бинарных классификаторов. Классификатор с номером *i* обучается следующим образом. Все объекты обучающей выборки делятся на два множества. В первое множество попадают объекты

Таблица 2

Комбинации параметров, которые были взяты при извлечении признакововых описаний сообщений

Параметр	Значение
Диапазон *n*-грамм (*n*1, *n*2)	$\{(n1, n2)\}_{2 \leqslant n1 \leqslant n2 \leqslant 12}$
Тип *n*-грамм (analyz)	Буквенные (char), словесные (word)
Dfmin	1, 2, 3
Тип признака	Бинарный, количественный
Лемматизация (lem)	Да, нет

[1] Список терминалов: https://tech.yandex.ru/tomita/doc/dg/concept/terminals-list-docpage/

с меткой i-го класса, а во второе множество — все остальные. После чего каждому объекту из первого множества приписывается метка 0, а каждому объекту из второго класса — метка 1. Таким образом получается новая обучающая выборка с прежними объектами, но новыми метками классов. На полученной выборке обучается бинарный классификатор. Если i-му классификатору подать на вход объект, для которого необходимо предсказать метку класса, то классификатор выдаст две оценки принадлежности: оценку принадлежности к классу i и оценку принадлежности к остальным классам без i-го. Обозначим оценку принадлежности к первому множеству, которую выдал i-й классификатор, через b_i.

Таким образом, если подать объект x k классификаторам, то получится k чисел b_1, ..., b_k. Теперь нужно выбрать из этих чисел максимальное и рассмотреть его номер. Этот номер и будет номером класса, к которому будет отнесен объект x.

При использовании стратегии ovo для обобщения бинарного классификатора на k классов создается $k(k-1)/2$ бинарных классификаторов. Если имеется k классов, то они образуют $k(k-1)/2$ пар. Для каждой полученной пары обучается классификатор. Рассмотрим процесс обучения классификатора, соответствующего паре, которую образуют i-й и j-й классы. Из обучающей выборки выделяются все объекты, имеющие метку класса с номером i, и все объекты с номером класса j. Выделенные объекты разбиваются на два множества и образуют новую обучающую выборку, в которой присутствует только два класса. На этой выборке обучается классификатор с номером (i, j). Если рассматриваемый алгоритм бинарной классификации выдает оценку принадлежности к классу, тогда при запуске классификатора с номером (i, j) будет получено две оценки: b_{ij} и b_{ji}. Запустив классификатор, соответствующий каждой такой паре, будет получено $k(k-1)/2$ пар оценок. Рассмотрим числа $b_i = \Sigma_{j' = i} b_{ij}$, где индекс $i = 1, ..., k$. Как и в ovr выбирается максимальное из $\{b_i\}$, и затем берется его номер. Это и будет номер класса, к которому будет отнесен объект x. Если классификатор не способен выдавать оценку принадлежности, а может выдавать только метку класса, то после запуска всех классификаторов будет получено $k(k-1)/2$ меток. Затем проводится подсчет меток, и выбирается метка, которая упоминается наибольшее число раз. К классу с этой меткой будет отнесен объект x. Отметим также, что классификация с подсчетом меток может породить сложную ситуацию, когда две и более меток набрали одинаковое максимальное число голосов.

В методе опорных векторов имеется функционал качества. В функционале качества содержится ряд параметров, которые можно изменять и, тем самым, влиять на качество обучения. Функционал качества имеет вид

$$L(w, w_0) = \sum_{i=1}^{l} g\left(1 - M_i(w, w_0)\right) + \frac{1}{2C}\|w\|^2.$$

Число $M_i(w, w_0) = (< x_i, w > - w_0)y_i$ называется отступом (margin). Операция $< u, v >$ обозначает скалярное произведение векторов u и v, w_0 является скалярной величиной.

Рассмотрим функцию g. На функцию g накладывается одно обязательное условие: $\max(z, 0) < = g(z)$. Поскольку на этапе обучения функционал качества будет подвергаться минимизации по w, w_0, то для того чтобы этот функционал можно было минимизировать стандартными методами минимизации (например, методом наискорейшего градиентного спуска), от функции g требуется гладкость или непрерывность. Функцию g называют функцией потерь. Часто в качестве функции потерь выбирают функцию hinge или squared hinge (приведены оригинальные названия функций).

При минимизации функционала качества норма параметра w может стать очень большой, что будет плохо сказываться на качестве классификации. Для предотвращения увеличения нормы параметра w добавляют штрафное слагаемое $\frac{\|w\|^2}{2C}$. Описанное штрафование за увеличение нормы вектора w называют регуляризацией, а штрафное слагаемое — регуляризатором. Числовой параметр C отвечает за размер штрафа. Обычно в качестве нормы выбирают l_2- или l_1-норму. Таким образом, алгоритм позволяет задать параметр C и выбрать норму. Как будет отмечено далее, в рамках данной работы проводили эксперименты с разными комбинациями параметров. В проделанных экспериментах регуляризация с нормой l_1 показывала худшие результаты, чем с l_2-нормой. В FAQ https://www.csie.ntu.edu.tw/~cjlin/liblinear/FAQ.html#l1_regularized_classification сказано про аналогичную ситуацию относительно l_2- и l_1-норм.

Существуют разные способы минимизации функционала качества. В частности, библиотека liblinear, которая реализует метод опорных векторов и используется в предлагаемом авторами подходе, представляет различные способы минимизации, в зависимости от выбранной нормы при регуляризации и стратегии обобщения. Опишем разные способы минимизации и условия, при которых эти способы можно использовать (см. рис. 9). Библиотека liblinear допускает две стратегии обобщения на многоклассовую классификацию — crammer-singer и one-vs-rest. Первая стратегия не позволяет выбирать ни норму, ни способ минимизации, ни функцию потерь. Вторая стратегия позволяет выбирать функцию потерь между hinge и squared hinge (см. рис. 9). В случае с hinge допускается только l_2-норма. В случае с функцией squared hinge допускается выбор норм между l_2- и l_1-нормами. Наконец, рассмотрим различные способы минимизации. Если в слагаемом-регуляризаторе используется l_2-норма, а в качестве функции потерь берется squared hinge, то функционал является гладким и его можно минимизировать, используя, например, метод градиентного спуска. Когда функционал качества минимизируется явно (без применения теоремы Каруша—Куна—Таккера), то говорят, что решается прямая задача. Вместе с тем можно воспользоваться и теоремой Каруша—Куна—Таккера и решать двойственную задачу.

Значение dual = *T* (см. рис. 9) означает, что решается двойственная задача, dual = *F* означает решение прямой задачи минимизации. В случае, когда в слагаемом-регуляризаторе используется l_1-норма и в качестве функции потерь используется squared hinge, то может применяться метод минимизации, именуемый LASSO [11].

Из отмеченных параметров, для решения поставленной задачи рассматривались комбинации, представленные на рис. 9, а также рассматривалось варьирование параметра *C*.

Время минимизации функционала качества с l_1-регуляризацией занимает значительно больше времени, чем с l_2-регуляризацией. В связи с этим обстоятельством в данной работе не проверялось, какое качество дает l_1 на всех конфигурациях параметров. Однако на всех комбинациях параметров с использованием l_1-нормы, на которых удалось измерить качество, l_1 давало худшее качество, чем l_2.

Несколько слов о параметре *C* из функционала качества. С l_2-регуляризатором этот параметр практически не влияет на качество. В случае с l_1 он оказывает сильное влияние. Поэтому, когда использовалась l_2-регуляризация, параметр *C* не менялся и всегда полагался равным 1.

Существуют ситуации, когда показания метрик качества могут ввести в заблуждение. Например, если алгоритм будет тестироваться на несбалансированной выборке, в которой 95 объектов принадлежат классу 0 и 5 объектов классу 1, то можно рассмотреть алгоритм, который все отнесет к нулевому классу. В рассмотренном примере будет получена точность 95 %, которая будет казаться хорошим результатом.

В связи с этим обстоятельством рассматривают алгоритмы, которые относят все объекты к одному классу. Такие алгоритмы называют константными. Используя константный алгоритм, проводят классификацию: отнесение всех объектов к одному классу. Затем вычисляют значения метрик качества. Далее полученные метрики качества у неконстантного алгоритма можно сравнивать с метриками константного алгоритма и делать суждения о том, насколько хорошо алгоритм проводит классификацию. В табл. 3 приведены измерения четырех метрик качества для трех константных алгоритмов: когда все объекты относятся к классу с меткой "–1", к классу с меткой "+1" и к классу с меткой "0". В столбце с названием "размер класса" указано, сколько процентов приходится на соответствующий класс в обучающей выборке.

Таблица 3

Метрики качества для константных алгоритмов

Тональность (класс)	Размер класса, %	Метрика			
		Macro precision	Macro recall	Macro F1	Accuracy
Отрицательная	26,35	0,088	0,33	0,14	0,26
Положительная	39,05	0,13	0,33	0,19	0,39
Нейтральная	34,60	0,12	0,33	0,17	0,35

В процессе экспериментов было проведено несколько тысяч измерений и составлен список топ 10 по качеству получаемых метрик. Использовались следующие метрики качества: accuracy, macro precision, macro recall, macro F1. Измерения метрик качества проводили с помощью кросс-валидации по пяти блокам. Это значит, что имеющийся корпус разбивали на пять частей, после чего проводили пять измерений качества. Один блок использовался для контроля качества, оставшиеся четыре — для обучения классификатора. После процесса обучения классификатор предсказывал метки классов в контрольном блоке. Далее, на основе предсказанных и реальных меток в контрольном блоке вычислялись метрики качества. Затем другой блок использовался в качестве контрольного, а четыре оставшихся — для обучения классификатора. Далее, в следующих итерациях каждый из блоков выбирался в качестве контрольного. В результате описанных действий получилось пять измерений каждой метрики качества. Затем проводили операцию усреднения. К пяти измерениям каждой метрики качества применяли операцию среднего арифметического. Результат описанных измерений представлен на рис. 10—13.

Опишем структуру рис. 10—13. Первые два столбца указывают диапазон *n*-грамм, которые извлекались из обучающей выборки. Например, *n*1 = 6 и *n*2 = 8 будет означать, что извлекались *n*-граммы длинами 6, 7, 8.

Следующие четыре столбца ac, mf, mp, mr иллюстрируют метрики качества: ac — accuracy; mf — macro F1 measure; mp — macro precision; mr — macro recall. Топ 10 делался по данным из каждого столбца с метрикой.

Столбец bin свидетельствует о том, какого типа использовались признаковые описания *n*-грамм. Если в столбце bin стоит значение T, это означает, что использовался бинарный признак. То есть если в сообщении встретилась *i*-я *n*-грамма, тогда на *i*-м месте признакового описания будет стоять 1, и 0 — в противном случае. Если в этом столбце стоит значение F, то это свидетельствует о том, что использовались количественные признаковые описания. Это означает, что учитывалась кратность появления *n*-грамм. Если *i*-я *n*-грамма встречалась *p* раз, то на *i*-е место признакового описания ставилось число *p*.

Столбец lem содержит информацию о том, проводилась ли лемматизация. Если в ячейке находится значение T, то это означает, что лемматизация проводилась. Если в ячейке находится значение F, то это означает, что лемматизация не проводилась.

В ячейках столбца dfmin могут располагаться числа 1, 2, 3. Если в этой ячейке стоит значение *k*, то это означает, что если *n*-грамма встретилась менее чем *k* раз в обучающем корпусе, то данная *n*-грамма не попадет в словарь *n*-грамм. В частности, значение dfmin = 1 означает, что если *n*-грамма встретилась, то она гарантированно попадет в словарь *n*-грамм.

В столбце analyz указан тип *n*-грамм. В этом столбце могут быть значения char и word. Если в ячейке стоит значение char, то это означает, что извлекались буквенные *n*-граммы. Если в ячейке стоит значение word, то это говорит о том, что извлекались словесные *n*-граммы.

В столбце dual содержится информация о том, решалась ли прямая или двойственная задача минимизации. Значение T означает, что решалась двойственная задача, значение F указывает на то, что решалась прямая задача. Значение None означает, что использовался алгоритм минимизации, предписанный стратегией обобщения бинарного классификатора на многоклассовую классификацию. Значение None встречается при использовании стратегии обобщения crammer-singer.

В столбце loss указано, какая использовалась функция потерь. Значение None означает то, что использовалась функция потерь, предписанная данной стратегией обобщения. В случае использования стратегии обобщения ovr допускались функции hinge и sqaured hinge.

Столбец mclass может содержать два значения: cr или ovr. Данный столбец указывает обобщение бинарной классификации на трехклассовую классификацию. Значение cr означает crammer-singer. Значение ovr означает one-vs-rest.

Столбец pen содержит информацию о том, какая норма использовалась в слагаемом-регуляризаторе. Допускались значения l_1 и l_2. Значение None означает, что норма предписывалась стратегией обобщения бинарной классификации на многоклассовую классификацию.

Все проделанные измерения метрик качества при разных вариациях параметров можно увидеть по адресу https://github.com/dzabraev/ton-analysis/blob/master/report/tables/measurment.tsv. Из данных, представленных на рис. 10—13, видно, что наилучший результат дают буквенные 6, 7, 8-граммы, причем параметр dfmin взят равным 1; а признаки рассматривались как бинарные.

n1	n2	ac	mf	mp	**mr**	bin	lem	dfmin	analyz	dual	loss	mclass	pen
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	sqhi	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	sqhi	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	F	1	char	None	None	cr	None
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	F	1	char	T	sqhi	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	hinge	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	sqhi	ovr	l2

Рис. 10. Топ 10 по macro F1

n1	n2	**ac**	mf	mp	mr	bin	lem	dfmin	analyz	dual	loss	mclass	pen
6	8	0.7	0.7	0.71	0.7	T	T	1	char	None	None	cr	None
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	sqhi	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	F	1	char	None	None	cr	None
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	sqhi	ovr	l2
6	8	0.7	0.69	0.71	0.7	T	T	1	char	T	sqhi	ovr	l2
6	8	0.7	0.69	0.71	0.7	T	F	1	char	T	sqhi	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	F	1	char	T	sqhi	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	hinge	ovr	l2

Рис. 11. Топ 10 по accuracy

n1	n2	ac	mf	**mp**	mr	bin	lem	dfmin	analyz	dual	loss	mclass	pen
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	hinge	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	F	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	sqhi	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	F	1	char	T	hinge	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	F	1	char	T	sqhi	ovr	l2
8	9	0.69	0.69	0.72	0.69	T	T	1	char	T	hinge	ovr	l2
8	9	0.69	0.69	0.72	0.69	T	T	1	char	T	sqhi	ovr	l2
8	9	0.69	0.69	0.72	0.69	T	T	1	char	None	None	cr	None
8	9	0.69	0.69	0.72	0.69	T	F	1	char	T	hinge	ovr	l2

Рис. 12. Топ 10 по macro precision

n1	n2	ac	mf	mp	**mr**	bin	lem	dfmin	analyz	dual	loss	mclass	pen
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	sqhi	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	sqhi	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	F	1	char	None	None	cr	None
6	8	0.7	0.7	0.71	0.7	T	F	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	T	hinge	ovr	l2
6	8	0.7	0.7	0.71	0.7	T	T	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	F	1	char	T	sqhi	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	hinge	ovr	l2
3	12	0.69	0.69	0.72	0.7	T	T	1	char	None	None	cr	None
3	12	0.69	0.69	0.72	0.7	T	T	1	char	T	sqhi	ovr	l2

Рис. 13. Топ 10 по macro recall

5. Статистика по прессе

В настоящем разделе представлены результаты использования разработанного классификатора применительно к текстам в некоторых СМИ. Анализу подверглись сообщения таких СМИ, как РИА Новости (http://ria.ru), ТАСС (http://tass.ru), Московский Комсомолец — MK (http://mk.ru), RT (https://russian.rt.com). С указанных сайтов было извлечено практически 100 % статей. Общее число статей по отдельным агентствам представлено в табл. 4

В качестве обучающей выборки экспертом были размечены 318 статей, опубликованных на сайте газеты "Московский Комсомолец". На этой выборке был обучен классификатор. Далее, с использованием разработанного классификатора были предсказаны метки классов для остальных статей.

Приведем иллюстрацию полученной статистики по СМИ. На рис. 14 (см. вто-

Таблица 4

Объем публикаций по отдельным агентствам

Информационное агентство	Общее число статей	Число статей с упоминанием МГУ
РИА	1,5 млн	5536
ТАСС	440 тыс.	836
МК	250 тыс.	318
RT	120 тыс.	134

рую сторону обложки) приведены распределения статей, которые улучшают репутацию МГУ (зеленый цвет), ухудшают репутацию МГУ (красный цвет) и те, в которых есть упоминание об МГУ, но они на репутацию никак не влияют (синий цвет). Для каждого столбика приведено процентное соотношение — какой процент данного класса встретился в СМИ за этот год. Справа от столбика указано число статей, которые попали в данный класс за год.

Гистограмму на рис. 14 можно детализировать по агентствам и представить в виде таблицы (табл. 5).

В каждой ячейке табл. 5 приведены три числа. Число, перед которым стоит плюс, означает, что данное агентство за данный год опубликовало данное число статей, которые улучшают репутацию МГУ. Минус стоит перед числом статей, ухудшающих репутацию. Самое нижнее число, перед которым ничего не стоит, означает число статей, которые не влияют на репутацию МГУ.

На рис. 15, 16 (см. третью сторону обложки) приведены гистограммы с детализацией по месяцам. Отметим, что в июне 2015 г. наблюдался всплеск отрицательных статей. Это связано с тем, что в прессе было много упоминаний о студентке МГУ, которая намеревалась вступить в ИГИЛ.

Таблица 5

Детализация распределения статей по агентствам

Агентство	2009 г.	2010 г.	2011 г.	2012 г.	2013 г.	2014 г.	2015 г.	2016 г.
RT	+0	+0	+0	+1	+18	+27	+19	+10
	−0	−0	−0	−0	−1	−1	−37	−3
	0	0	0	1	4	5	4	2
МК	+18	+22	+16	+9	+14	+16	+14	+2
	−9	−8	−10	−12	−13	−4	−25	−2
	25	14	26	28	14	8	8	0
РИА	+0	+186	+717	+684	+804	+685	+825	+257
	−0	−4	−31	−40	−57	−1	−156	−24
	0	45	259	263	270	112	96	36
ТАСС	+0	+0	+14	+78	+136	+57	+177	+75
	−8	−8	−2	−6	−29	−4	−47	−13
	0	0	9	32	63	39	37	17

6. Направления дальнейших исследований

К направлениям дальнейших исследований в рамках поставленной задачи и с позиции использования результатов ее решения на практике можно отнести следующие.

- Увеличение точности классификации. На этом направлении планируется повысить точность классификации за счет использования более современных алгоритмов классификации на основе механизмов искусственных нейронных сетей.
- Извлечение информации о цитировании в СМИ сотрудников МГУ и результатов их деятельности, а также сведений о выступлениях в СМИ представителей МГУ. Эту задачу предполагается решать с помощью составления контекстно-свободных грамматик и использования программы tomita-parser.
- Повышение производительности собственно анализатора и процессов анализа новостных событий в СМИ в реальном масштабе времени. Представляется необходимым усовершенствовать систему сбора и обработки данных в целях оперативной обработки новых публикаций. Это позволит сократить интервал времени между появлением критически важной информации в СМИ и соответствующей реакцией на нее со стороны администрации МГУ.

Заключение

Представлены результаты исследований, направленных на поиск эффективных методов и средств автоматизации процессов поиска, анализа текстовых сообщений в СМИ, имеющих отношение к отдельным субъектам различного рода деятельности, а также процессов оценки их тональности. Кратко изложены результаты анализа функциональных возможностей уже существующих моделей, методов и программных механизмов, которые могут быть использованы для решения поставленной задачи.

Описаны разработанные на основе такого анализа программные средства извлечения текстовых сообщений в СМИ и оценки их тональности. Продемонстрированы результаты их тестирования на примере сообщений в нескольких крупнейших СМИ за последние пять лет применительно к МГУ имени М. В. Ломоносова. Они свидетельствуют о практической реализуемости и хороших перспективах применяемых в их составе моделей и методов.

Список литературы

1. **Интеллектуальная** система тематического исследования научно-технической информации (ИСТИНА) /Под ред. акад. А. Садовничего. М.: Изд-во Моск. гос. ун-та, 2014. 262 с.
2. **Воронцов К. В.** Математические методы обучения по прецедентам. URL: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf
3. **Spärck Jones K.** A Statistical Interpretation of Term Specificity and Its Application in Retrieval // Journal of Documentation. 1972. Vol. 28, N. 1 P. 11—21.
4. **Martineau J., Finin T.** Delta TFIDF: An Improved Feature Space for Sentiment Analysis. URL: http://ebiquity.umbc.edu/_file_directory_/papers/446.pdf
5. **Усталов Д. А**. Извлечение терминов из русскоязычных текстов при помощи графовых моделей. URL: http://koost.eveel.ru/science/CSEDays2012.pdf

6. **Kalchbrenner N., Grefenstette E., Blunsom P.** A Convolutional Neural Network for Modelling Sentences. URL: http://arxiv.org/pdf/1404.2188v1.pdf
7. **Yoon Kim.** Convolutional Neural Networks for Sentence Classification, 2014. URL: http://arxiv.org/pdf/1408.5882v2.pdf
8. **Ганкин Г. М.** Определение эмоциональной окраски коротких текстовых сообщений // Программная инженерия. 2013. № 9. С. 33—41.
9. **Crammer K., Singer Y.** On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. URL: http://www.jmlr.org/papers/volume2/crammer01a/crammer01a.pdf
10. **Обобщение** бинарной классификации на многоклассовую классификацию. URL: https://en.wikipedia.org/wiki/Multiclass_classification
11. **Tibshirani R. J**. Regression shrinkage and selection via the lasso // Journal of the Royal Statistical Society. Series B (Methodological). 1996. Vol. 58, N. 1. P. 267—288.

Methods of Automated Sentiment Analysis of Texts Published by Mass Media

V. A. Vasenin, vasenin@msu.ru**, V. A. Roganov**, var@msu.ru**, M. D. Dzabraev**, dzabraew@gmail.com, Lomonosov Moscow State University, 119234, Moscow, Russian Federation

Corresponding author:

Vasenin Valery A., Professor, Lomonosov Moscow State University, 119234, Moscow, Russian Federation, e-mail: vasenin@msu.ru

Received on April 30, 2016
Accepted on May 20, 2016

This article examines approaches to solving the problem of the analysis of natural language texts to identify the emotional color in relation to a particular subject. After brief analysis of the known methods that can be used to solve this problem we describe the proposed approach and developed layout of the information service usable for tone analysis of publications in the mass media in relation to Moscow State University named after M. V. Lomonosov, which is large and significant subject of national scientific, technical and educational activities. A representative sample of publications has been used to test and analyze the proposed method. Developed methods and tools will be used as components of information services, dedicated to separate categories of users of information-analytical system (IAS) "ISTINA". This system is designed and developed under a separate project in the Moscow State University named after M. V. Lomonosov as one of the components of the overall system of university management. The main task which is solved with use of the "ISTINA" system is thematic analysis of scientometric data to help with preparation and adoption of managerial decisions.

Keywords: *text analysis, tone analyzer, mass media analysis, facts extraction, machine teaching, convolutional neural network*

For citation:

Vasenin V. A., Roganov V. A., **Dzabraev M. D.** Methods of Automated Sentiment Analysis of Texts Published by Mass Media, *Programmnaya Ingeneria,* 2016, vol. 7, no 8, pp. 360—372.

DOI: 10.17587/prin.7.360-372

References

1. **Sadovnichiy V. A., Afonin S.A, Bakhtin A. V.** et al. *Intellektual'naja sistema tematicheskogo issledovanija nauchno-tehnicheskoj informacii (ISTINA)* (Intellectual System of Thematic Examination of Scientific and Technical Data (ISTINA)), Moscow, Moscow State University Publishing, 2014, 262 p. (in Russian).
2. **Voroncov K. V.** Matematicheskie metody obuchenija po precedentam (Mathematical methods of learning by precedents), available at: http://www.machinelearning.ru/wiki/images/6/6d/Voron-ML-1.pdf (in Russian).
3. **Sparck Jones K**. A Statistical Interpretation of Term Specificity and Its Application in Retrieval, *Journal of Documentation,* 1972, vol. 28, no. 1, pp. 11—21.
4. **Martineau J., Finin T.** Delta TFIDF: An Improved Feature Space for Sentiment Analysis, available at: http://ebiquity.umbc.edu/_file_directory_/papers/446.pdf
5. **Ustalov D. A.** Izvlechenie terminov iz russkojazychnyh tekstov pri pomoshhi grafovyh modelej (Term extraction from texts in Russian language with the help of graph models), available at: http://koost.eveel.ru/science/CSEDays2012.pdf (in Russian).
6. **Kalchbrenner N., Grefenstette E., Blunsom P.** A Convolutional Neural Network for Modelling Sentences, available at: http://arxiv.org/pdf/1404.2188v1.pdf
7. **Yoon Kim.** Convolutional Neural Networks for Sentence Classification, available at: http://arxiv.org/pdf/1408.5882v2.pdf
8. **Gankin G. M.** Opredelenie jemocional'noj okraski korotkih tekstovyh soobshhenij (Recognizing Sentiments of Short Text Messages), *Programmnaya Ingeneria,* 2013, no. 9, pp. 33—41 (in Russian).
9. **Crammer K., Singer Y.** On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines, available at: http://www.jmlr.org/papers/volume2/crammer01a/crammer01a.pdf
10. **Obobshhenie** binarnoj klassifikacii na mnogoklassovuju klassifikaciju (A generalization of binary classification to multiCLASS classification), available at: https://en.wikipedia.org/wiki/Multiclass_classification (in Russian).
11. **Tibshirani R. J**. Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society. Series B (Methodological),* 1996, vol. 58, no. 1, pp. 267—288.

УДК 629.78 DOI: 10.17587/prin.7.373-382

Н. В. Рябогин, нач. отделения, e-mail: n.ryabogin@mokb-mars.ru,
М. А. Шатский, канд. техн. наук, нач. направления, **М. Ю. Косинский**, канд. техн. наук, нач. отдела, **В. Н. Соколов**, д-р техн. наук, зам. дир. по научной работе, зам. ген. конструктора, ФГУП Московское опытно-конструкторское бюро "Марс", г. Москва,
Н. М. Задорожная, канд. техн. наук., доц., МГТУ имени Н. Э. Баумана

Применение языка SysML в задачах разработки и отработки программного обеспечения бортовых комплексов управления космическими аппаратами

Рассмотрено применение языка SysML для повышения эффективности процесса разработки и описания функционирования программного обеспечения бортового комплекса управления космическим аппаратом различного назначения. Целью применения SysML является повышение качества и надежности процесса разработки и отработки программного обеспечения бортового комплекса управления космическим аппаратом через внедрение современных средств моделирования и информационных технологий.

Ключевые слова: *системный анализ, графические средства системного анализа, программное обеспечение, бортовые комплексы управления, космические аппараты*

Введение

Бортовой комплекс управления космического аппарата (БКУ КА) представляет собой сложный многокомпонентный комплекс, содержащий в себе как аппаратные, так и программные средства [1—3]. Одним из примеров БКУ КА может служить бортовая система управления (БСУ) КА "Спектр-Р", структурная схема которой представлена на рис. 1. Как видно на схеме, в состав БКУ кроме измерительных и исполнительных органов, входят семь блоков управления.

Основной целью при проектировании архитектуры бортового программного обеспечения (БПО) является создание иерархической структуры БПО, включающей системы, подсистемы и отдельные модули. Это является необходимым условием успешного создания любого сложного программного комплекса большой группой разработчиков.

В процессе создания БКУ КА не меньше половины времени и ресурсов уходит на разработку и отработку БПО. Отработка БПО БКУ КА занимает больше 70 % времени его создания. Сам процесс разработки и отработки описан много раз и, несмотря на это, остается трудоемким и для больших комплексов слабо наблюдаемым.

Описание программного обеспечения (ПО), согласно ГОСТ, руководящим документам и учебным пособиям, проводится несколькими способами: блок-схемами алгоритма с таблицами используемых переменных, текстовым описанием и текстом программы на используемом языке [3, 4]. Такой способ описания в первую очередь направлен на документирование ПО и обеспечение работы разработчика ПО. Существующие методы документирования ПО уделяют повышенное внимание вопросам программирования, при этом мало описывая архитектуру ПО и общую логику работы системы.

Однако даже при достаточно полном документировании ПО для больших комплексов ПО зачастую наступает момент, когда уже и разработчикам ПО сложно разобраться в существующем ПО, восстановить связи внутри ПО и провести необходимые корректировки.

Основными официально признанными методами описания ПО являются тексты программ, текстовое описание программ и блок-схемы алгоритмов. Очевидно, что перечисленные методы описания обладают условием необходимости, но не обладают условием достаточности. О недостаточности описания говорит хотя бы тот факт, что каждый раз при разработке, доработке и отладки любого ПО уходит большое количество времени и ресурсов на изучение функций, переходов и взаимодействия между отдельными частями кода.

Актуальность задачи отслеживания взаимодействия и переходов между отдельными частями ПО вызвана повсеместным применением модульного принципа во всех сферах техники, особенно в разработке ПО [3].

На каждом этапе разработки архитектуры БПО проводится выпуск соответствующей документации:

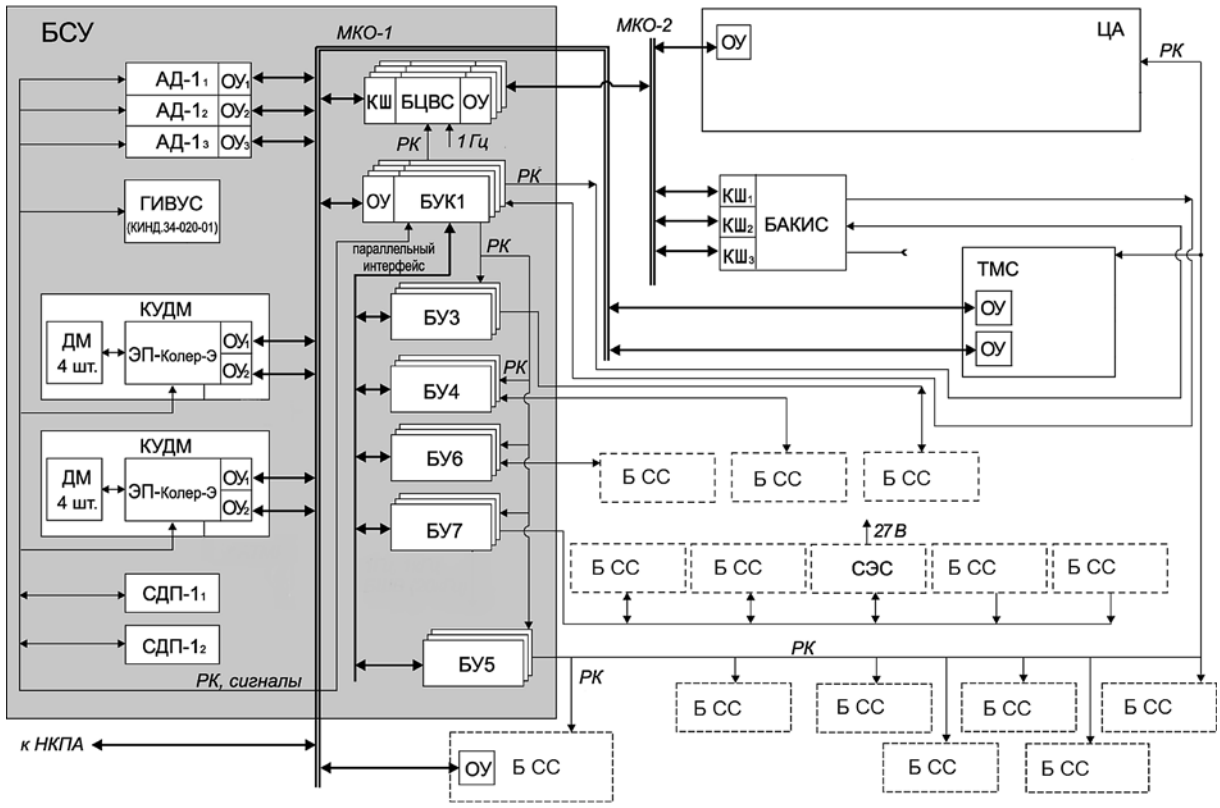


Рис. 1. Структурная схема БКУ КА "Спектр-Р":

БЦВС — бортовая цифровая вычислительная система; КШ — контроллер шины; ОУ — оконечное устройство; БУК1 — блок управления и контроля; БУ — блок управления; АД-1 — астродатчик; ГИВУС — гироскопический интегратор угловых скоростей; КУДМ — комплекс управляющих двигателей-маховиков; ДМ — двигатель-маховик; СДП-1 — солнечный датчик положения; ЦА — целевая аппаратура; БАКИС — бортовая аппаратура командно-измерительной станции; ТМС — телеметрическая система; Б СС — блок смежных систем; РК — разовая команда; НКПА — наземная контрольно-проверочная аппаратура; СЭС — система электроснабжения; МКО — манчестерский канал обмена; ЭП — электронный прибор

протоколов информационно-логического взаимодействия, различных перечней и других документов, чаще всего в текстовом или табличном виде, реже в виде схем, например, структурных. При этом известно, что для человеческого восприятия более понятным является именно графическое представление информации (особенно в случае сложных многокомпонентных систем) [5, 6]. Однако используемые графические средства, с одной стороны, должны быть понятны для всех, т. е. стандартизированы, а с другой стороны, отражать специфику задачи и быть наглядными и удобными, что часто входит в противоречие. Так, например, наиболее часто используемые блок-схемы алгоритмов Единой системы программной документации (ЕСПД) слабо приспособлены для рассматриваемой задачи. Альтернативой мог бы быть широко используемый при разработке прикладного ПО язык UnifiedModelingLanguage (UML), стандарт 1997 г., однако он во многом ориентирован именно на задачи программной реализации: объекты, интерфейс пользователя, что менее актуально для БПО.

Одним из методов улучшения процессов, связанных с разработкой и отработкой ПО, является применение современных графических средств системного анализа [5—8]. В настоящее время наблюдается расширение области задач разработки ПО, в которых применяются графические средства разработки ПО, в том числе и для применения в БКУ КА [9, 10].

Целью работы, результаты которой описаны в статье, явилась адаптация существующих графических средств описания сложных иерархических систем для дополнительного графического описания и документирования ПО БКУ в целях обеспечения наблюдаемости структуры, допустимых состояний и режимов работы, связей между подсистемами и соответствующего упрощения отладки системы.

Использование современных графических средств системного анализа

Наиболее подходящим для рассматриваемой задачи можно считать язык SystemModelingLanguage (SysML), развиваемый как расширение UML 2 в целях разработки, анализа и верификации сложных динамических систем. Графическая связь SysML с UML 2 изображена на рис 2. SysML является графическим языком моделирования, который реализует анализ, спецификацию, разработку и проверку сложных систем [7].

Язык SysML представляет собой набор из диаграмм различного назначения, объединенных в группы по

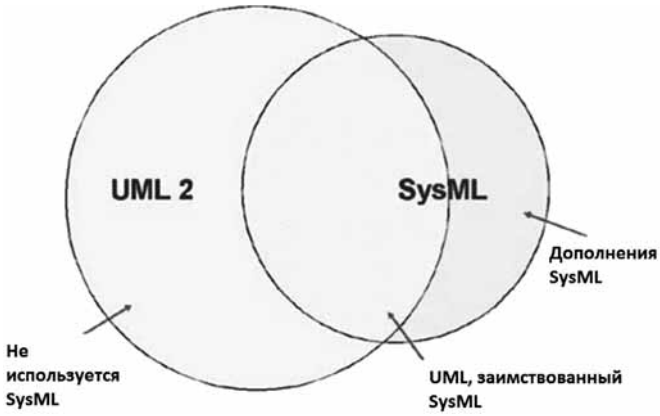


Рис. 2. Взаимосвязь SysML и UML 2

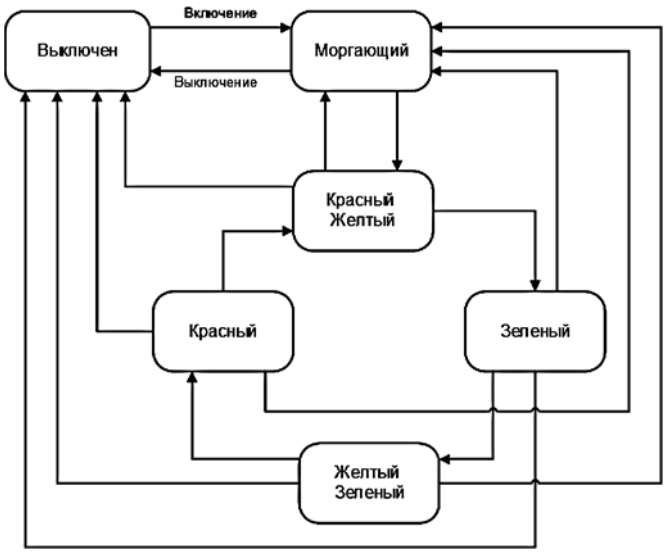


Рис. 3. Диаграмма состояний светофора в описании SysML

назначению: структурные, поведенческие (диаграммы деятельности, последовательности, состояния, вариантов использования), диаграммы требований.

Простейший пример диаграммы состояний изображен на рис. 3. Данная диаграмма отражает работу светофора, состояния, в которых находится светофор, и переходы между состояниями.

По результатам анализа стандарта, описывающего SysML, можно сделать вывод о том, что эти диаграммы практически полностью соответствуют по назначению сложившимся этапам разработки архитектуры ПО, хотя и имеют определенные отличия, как, впрочем, и любое новшество [8].

Диаграммы требований могут быть использованы для визуального представления текстовых требований технического задания (ТЗ) и их связей между собой (рис. 4). Для большего понимания диаграммы требований могут создаваться как для всей системы,

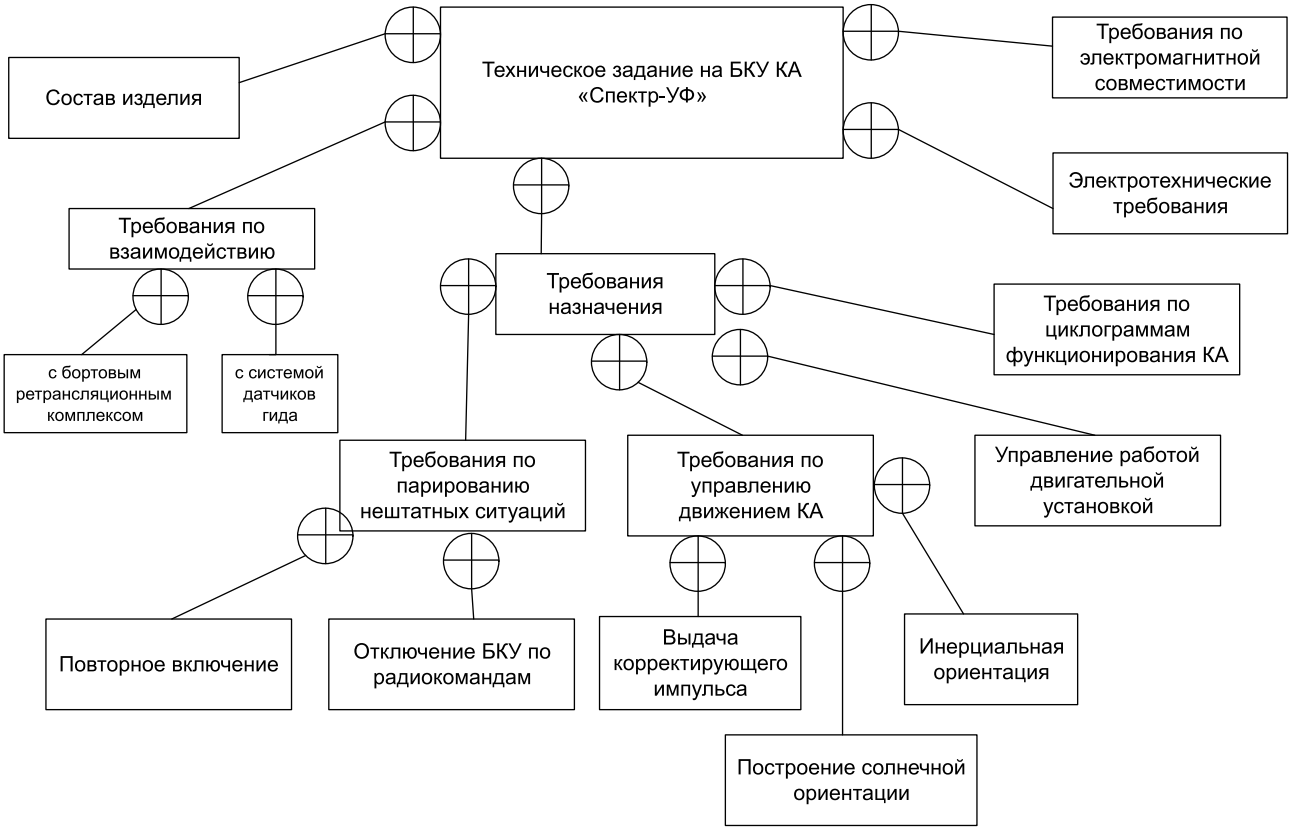


Рис. 4. Визуальное представление требований ТЗ с помощью диаграммы требований

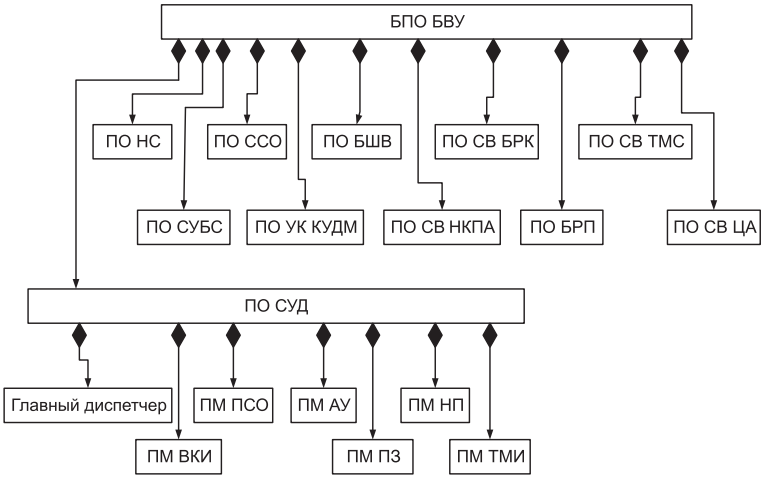


Рис. 5. Визуальное представление спецификации БПО с помощью структурной диаграммы:

БВУ — блок вычислительных устройств; НС — навигационная система; ССО — система стабилизации и ориентации; БШВ — бортовая шкала времени; СВ — система взаимодействия; БРК — бортовой ретрансляционный комплекс; СУБС — система управления бортовыми системами; УК — управление и контроль; БРП — блок распределения памяти; СУД — система управления движением; ПМ — программные модули; ПСО — построение солнечной ориентации; ВКИ — выдача корректирующего импульса; АУ — автономное управление; ПЗ — полетное задание; НП — настраиваемые параметры; ТМИ — телеметрическая информация

так и для отдельных ее частей и режимов работы.

Диаграммы, изображенные на рис. 4 и на следующих рисунках, содержат обозначения, присущие реальному БПО, и приведены только для иллюстрации работы предлагаемого подхода.

Структурные диаграммы могут быть использованы для описания спецификации БПО. На рис. 5 показано описание структуры БПО БКУ, полученное на основе анализа требований ТЗ [1].

Диаграммы состояний и действий SysML могут быть использованы для описания логики смены режимов работы КА и действий в нештатных ситуациях (НШС). Например, подобная ситуация приведена на рис. 6. На диаграмме, представленной на рис. 6, показаны возможные режимы работы системы управления движением КА, переходы между ними, а также условия переходов между режимами.

Выделение режимов работы системы управления (СУ) подобным образом позволяет разработчику или группе разработчиков реализовывать и отлаживать соответствующий каждому режиму набор алгоритмов автоном-

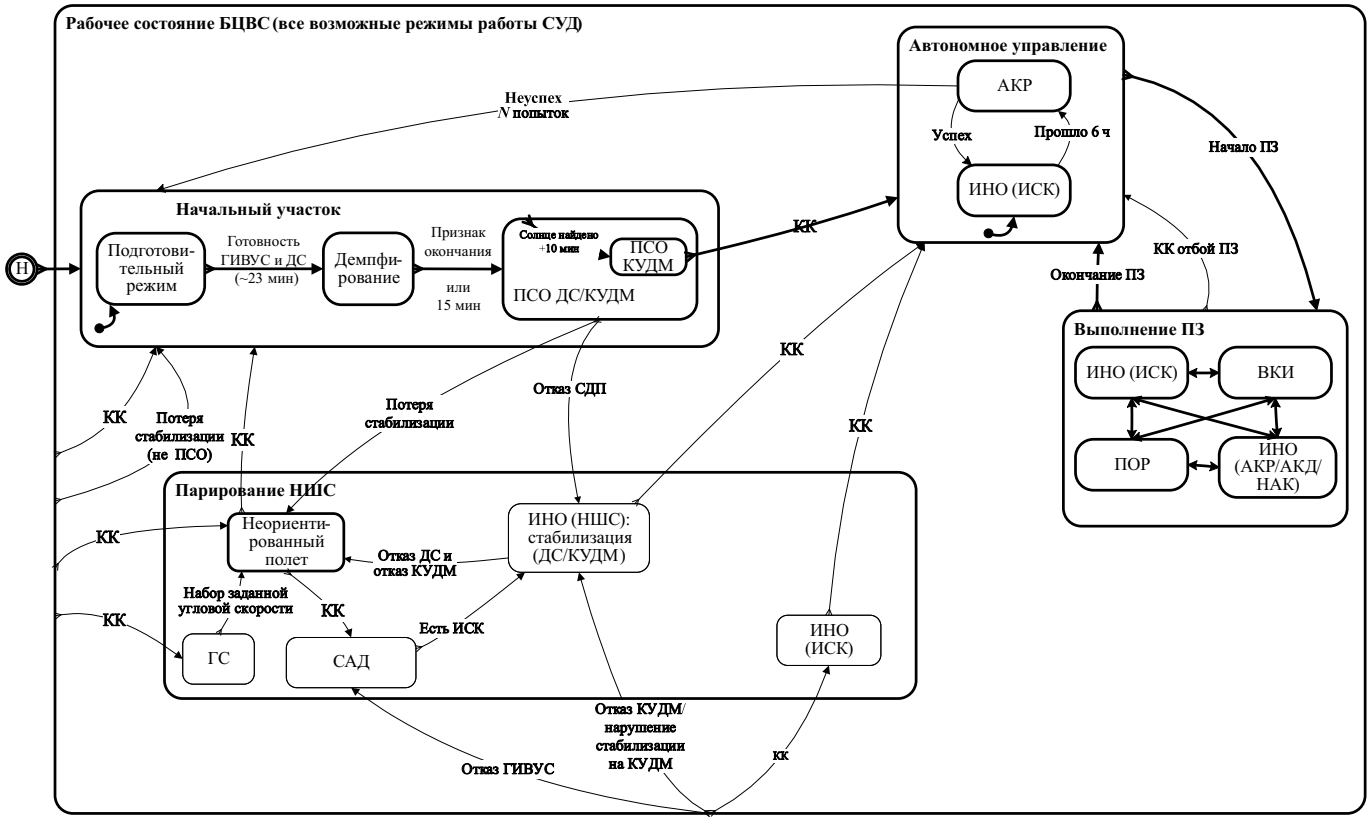


Рис. 6. Описание логики смены режимов с помощью диаграммы состояний и действий:

ДС — реактивные двигатели стабилизации; АКР — астрокоррекция ориентации; ИНО — инерциальная ориентация; ИСК — инерциальная система координат; АКД — астрокалибровка дрейфов; НАК — непрерывная астрокоррекция ориентации; ПОР — режим прецизионной ориентации; САД — стабилизация на астродатчиках; СДП — солнечный датчик положения; КК — кодовая команда; ГС — гиростабилизация

Рис. 7. Описание взаимодействия подсистем БПО с помощью диаграммы последовательности

но, используя упрощенные имитационные математические модели.

Диаграммы последовательности могут использоваться для описания протоколов взаимодействия подсистем и т. д. Например, процесс запуска одного из режимов работы КА представлен на рис. 7. Показано, что инициатором режима является подсистема СУД (система управления движением), по запросу которой смежные подсистемы — НС (навигационная система) и ССО (система стабилизации и ориентации) начинают подготовку оборудования и выставляют подтверждение его готовности. При поступлении признака завершения предыдущих операций от системы стабилизации и ориентации СУД окончательно переводит смежные системы в новый режим.

Предложенный в статье подход последовательно включает в себя несколько этапов и охватывает весь процесс разработки БПО. На начальном этапе основной задачей является выделение и группировка требований ТЗ. Затем на основе результатов анализа ТЗ разрабатывается иерархическая структура алгоритмов и выделяются автономные функции, выполнение которых требует минимального взаимодействия между подсистемами СУ. Определяются режимы работы подсистем, а также последовательность действий в каждой отдельной операции.

Применение диаграмм состояний в разработке программного обеспечения

Дополнительным методом описания функционирования ПО БКУ КА может быть описание через диаграммы состояний, что позволяет наглядно описывать структуру ПО, состояния ПО и определение переходов между этими состояниями. Предлагаемый метод позволяет наглядно описывать ключевые стадии и режимы работы ПО, адекватно отражать возможные переходы между режимами и значительно улучшает как разработку программ и методик испытаний, так и отработку ПО.

Представим несколько примеров применения предлагаемого метода. Метод заключается в выделении в ПОБ КУ отдельных состояний на разных уровнях функционирования и описании взаимодействия между этими уровнями и состояниями средствами и описаниями языка UML.

На высоком уровне при существовании операционного и специального программного обеспечения (ОПО и СПО соответственно) взаимодействие между ними внутри такта вычислительной машины может быть описано диаграммой, представленной на рис. 8.

Представленная на рис. 8 диаграмма отражает взаимодействие между ОПО и СПО и информационные потоки между ними. Каждая отдельная информационная связь выражается информационным протоколом взаимодействия.

Опускаясь на уровень ниже, метод применяется уже для описания состояний либо ОПО, либо СПО. На рис. 9 и 10 представлены диаграммы состояния верхнего уровня СПО, на котором описывается взаимодействие между группами модулей, отвечающих за режимы работы звездного прибора (ЗП).

Диаграмма, представленная на рис. 9, демонстрирует набор режимов функционирования типового ЗП и допустимые переходы между указанными режимами. Также на диаграмме приведен пример раскрытия сложного состояния: автономный режим работы представлен как совокупность режима определения ориентации и отслеживания ориентации и соответствующего взаимодействия между ними. Все обозначенные переходы по сути являются условными переходами, описываемыми протоколами информационного взаимодействия.

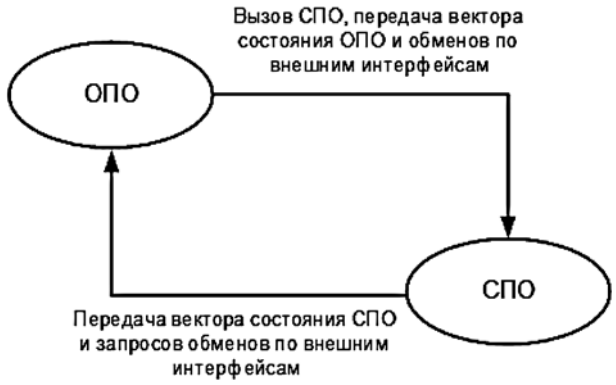


Рис. 8. Диаграмма взаимодействия ОПО и СПО

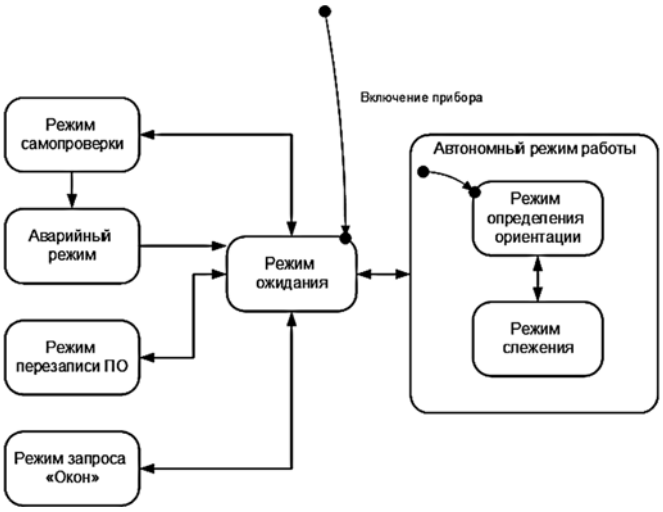


Рис. 9. Диаграмма режимов работы типового ЗП

На рис. 10 представлена частная диаграмма переходов между режимами ЗП. На диаграмме представлено взаимодействие между режимом ожидания и автономным режимом работы. Переходы представлены определенными условиями: переход из режима ожидания возможен только по команде. В случае возникновения нештатной ситуации, вызывающей срыв слежения или определения, реализуются обратные переходы из режима слежения вплоть до возврата в режим ожидания.

Примеры, представленные на рис. 9 и 10, демонстрируют применение SysML для описания взаимодействия сложных состояний, режимов работы, приборов и программных комплексов.

Первый шаг метода применения SysML заключается в определении описываемых состояний, определении уровня состояния и функций состояния. В примерах на рис. 9 и 10 состояния представлены

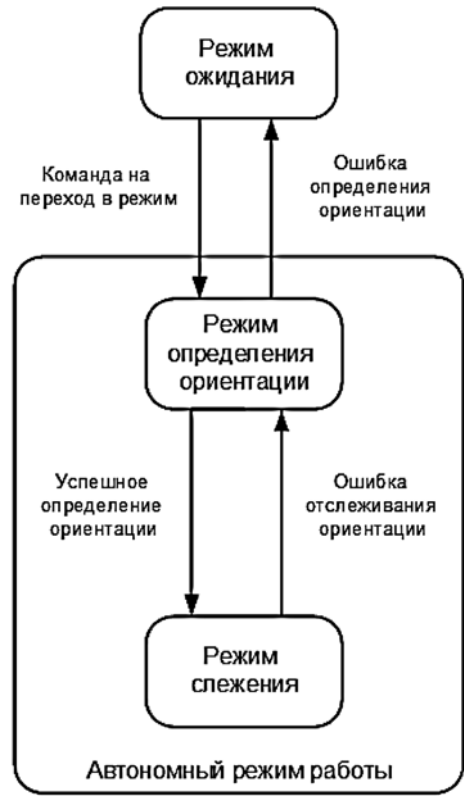


Рис. 10. Частная диаграмма перехода между режимами ЗП

определенными алгоритмами, отвечающими за конкретные функции системы.

Иным наполнением состояния может быть вектор признаков, определяющих работу функционально связанных групп алгоритмов. На высоком уровне таким примером служит диаграмма взаимодействия ОПО и СПО, представленная на рис. 8.

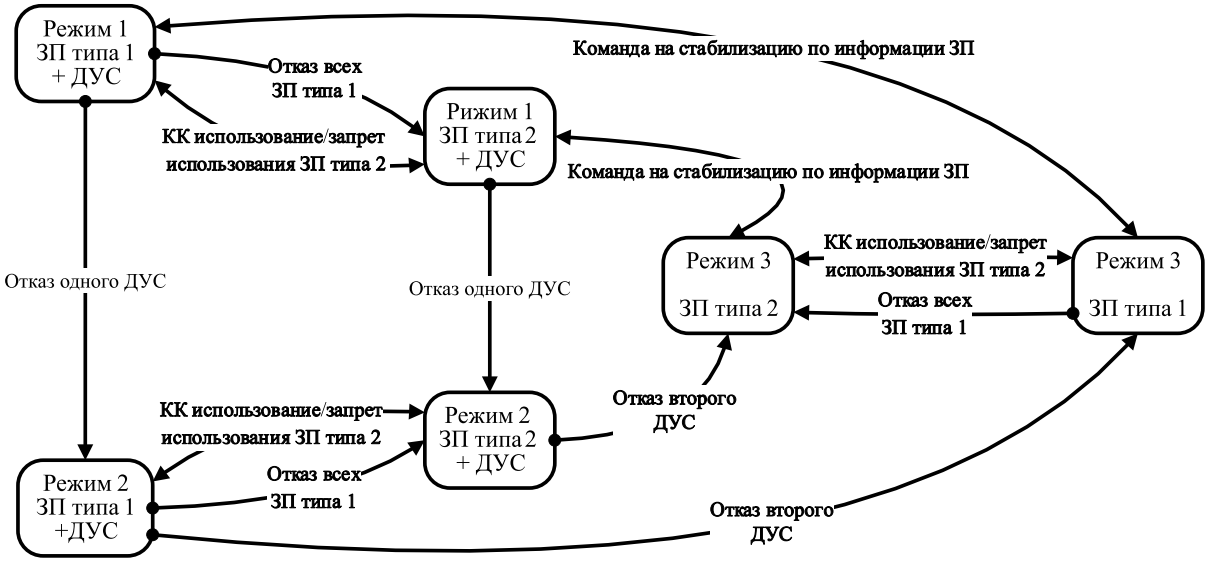


Рис. 11. Диаграмма переходов между режимами использования звездных датчиков в составе БКУ КА

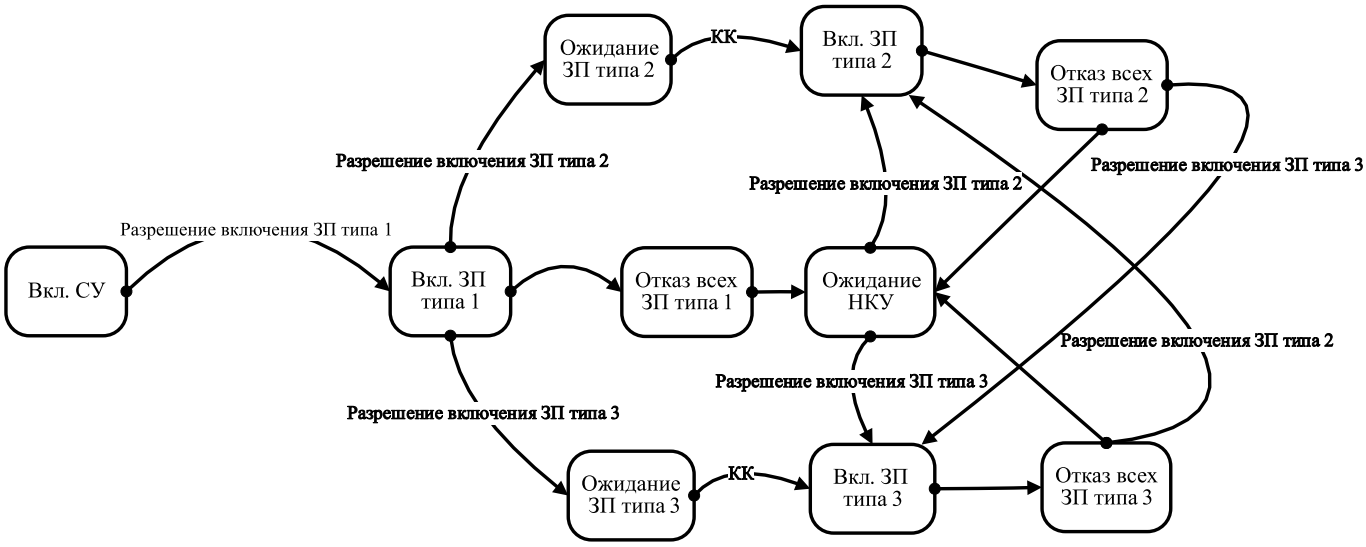


Рис. 12. Диаграмма переходов при задействовании резервов ЗП. Использованное сокращение: НКУ — наземный комплекс управления

На низких уровнях состояния могут отвечать за демонстрацию резервирования системы и применения резервов. К примеру, на КА установлено два типа ЗП. Число звездных приборов первого типа равно 2, второго типа — 1. Звездные приборы работают в режимах комплексирования информации с гироскопическими датчиками угловых скоростей (ДУС). Режимы комплексирования ДУС с ЗП разного типа отличаются в силу различий в работе приборов.

Описать переходы между этими режимами можно с помощью диаграммы, представленной на рис. 11. Данная диаграмма отображает работу программного комплекса, в который входят алгоритмы управления ЗП, алгоритмы управления ДУС, алгоритмы функционального контроля измерительной информации и алгоритмы комплексирования информации.

Пример описания применения резервов ЗП трех типов представлен на рис. 12. Данная диаграмма отображает состояние алгоритмов управления ЗП в зависимости от применяемого типа ЗП. Данная диаграмма отражает необходимые переходы между состояниями алгоритмов, при этом сами состояния алгоритмов выражаются конфигурацией признаковой информации.

Применение диаграмм состояния при анализе нештатных ситуаций

Функционирование КА связано с большим числом особенностей и ограничений, которые могут усложнить анализ возникновения и развития нештатных ситуаций. Одним из таких ограничений, к примеру, является канал сброса телеметрии. Работа телеметрического канала связана с ограничениями объема передаваемой информации и ее потерей.

Продемонстрируем применение диаграмм состояния для анализа нештатных ситуаций на примере ситуаций, связанных со сбоями и отказами в ГИВУС на космическом аппарате "Экспресс МД-1".

Функциональный тракт приема и обработки сигналов ГИВУС представлен на рис. 13. Прибор ГИВУС состоит из четырех независимых измерительных каналов (ИК) с аналоговым выходом. Выходная информация ГИВУС поступает на обработку в устройство преобразования, которое формирует информацию от ГИВУС в цифровом виде для алгоритмов расчета ориентации.

На этапе разработки БКУ осуществляется анализ видов и последствий критичности отказов, в ходе которого проводится выделение состояний БКУ при возникновении отказов. На диаграмме, представленной на рис. 14, отображены состояния БКУ и переходы между этими состояниями.

Указанная диаграмма применяется при анализе телеметрической информации с КА при возникновении нештатных ситуаций в измерительном тракте ГИВУС. Однозначность переходов позволяет восстанавливать причину отказа или перехода в отказное состояние, избежать возможного зацикливания алгоритмов. Диаграмма представляет состояния высокого уровня. Частный переход, представленный на более низком уровне, отображен на рис. 15.

Наглядность и информативность диаграммы, представленной на рис. 15, позволяют при анализе нештатных ситуаций по данным телеметрии использовать конечное число гипотез и проверок. При возможном отсутствии в телеметрии данных о срабатывании определенного типа контроля применение диаграммы состояний позволяет сократить время восстановления картины событий и развития отказа.

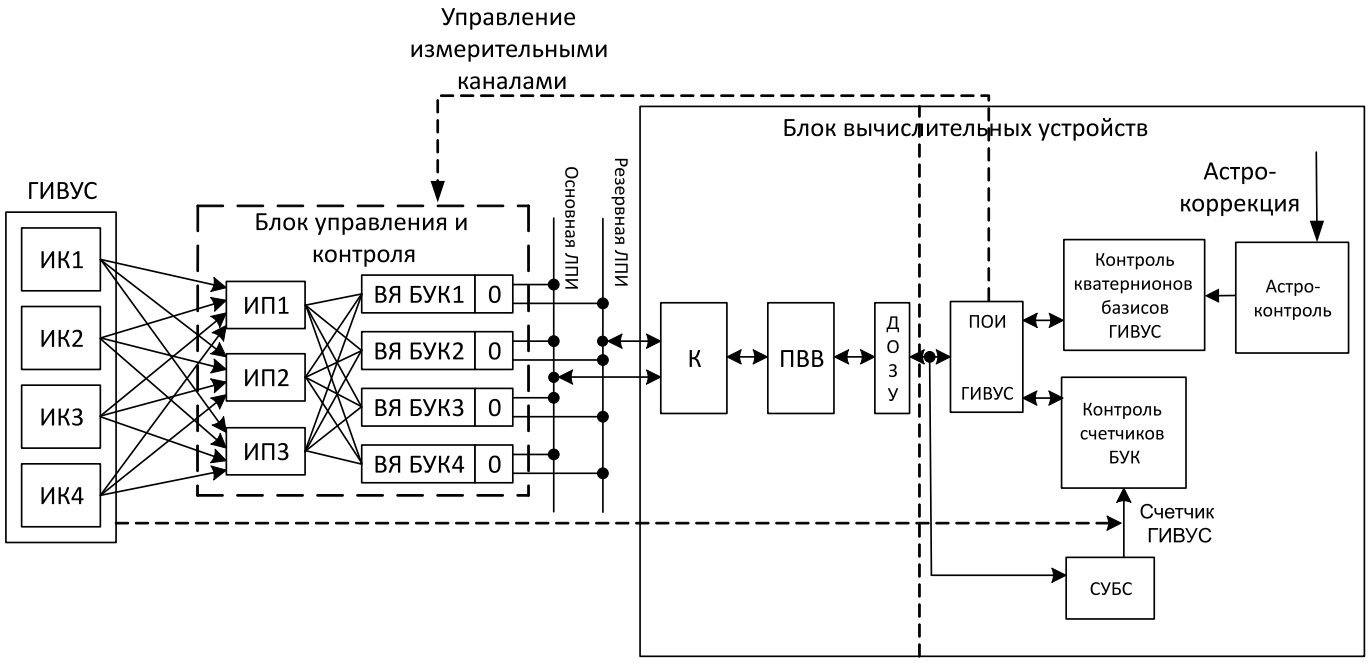


Рис. 13. Функциональный тракт ГИВУС:

ИП — исполнительная плата; ВЯ БУК — вычислительное ядро блока управления и контроля; О — оконечное устройство; ЛПИ — линия передачи информации; К — контроллер ЛПИ; ПВВ — процессор ввода-вывода; ДОЗУ — двунаправленное оперативное запоминающее устройство; ПОИ — предварительная обработка информации

Диаграмма состояний отказов БКУ позволяет не только наглядно и открыто разрабатывать программно-математическое обеспечение, но и в случае анализа нештатных ситуаций, возникающих при функционировании КА, быстро проводить анализ причин и сценариев развития отказов.

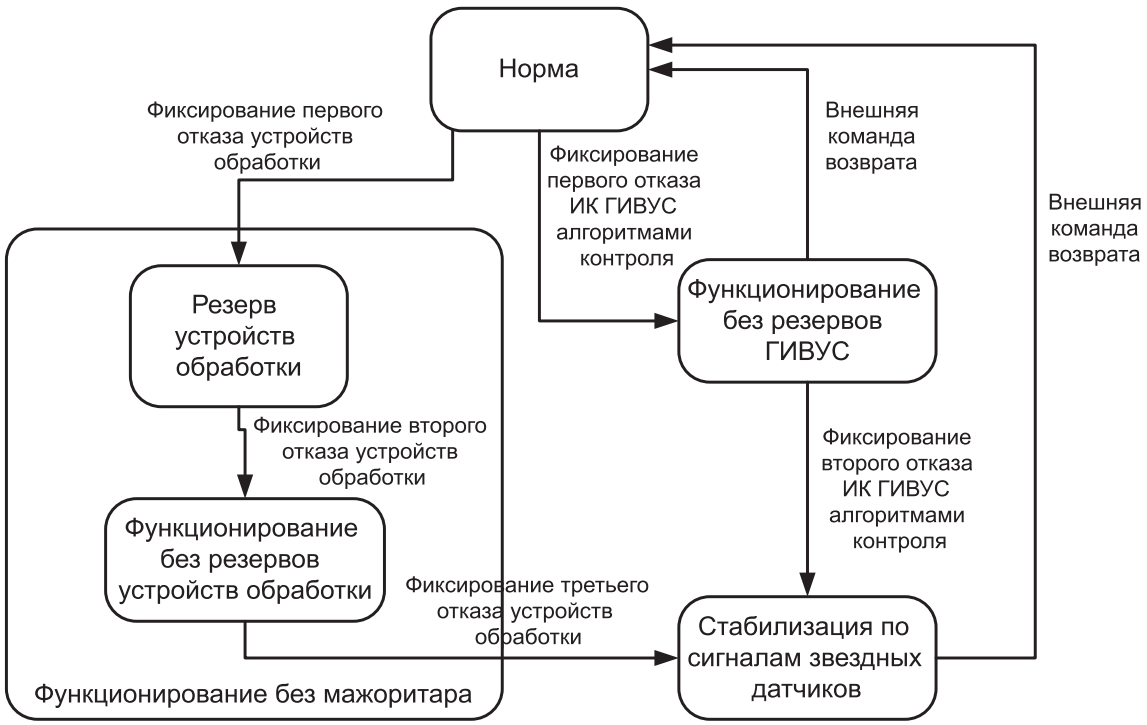


Рис. 14. Диаграмма состояний БКУ при отказах измерительного тракта ГИВУС

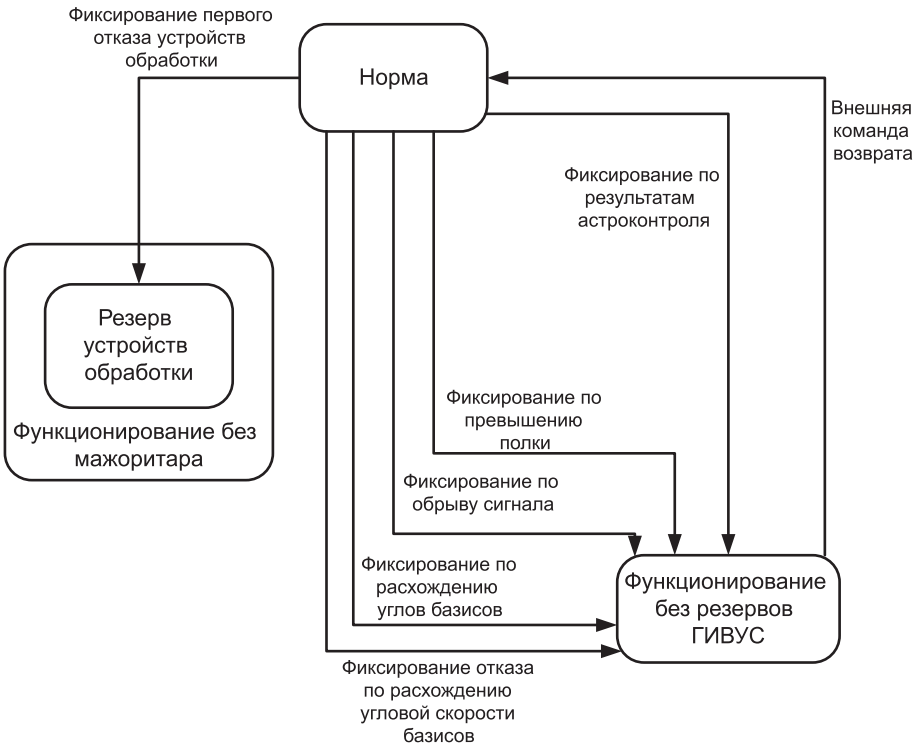


Рис. 15. Частная диаграмма переходов при фиксировании первых отказов

Заключение

Представлен метод описания ПО бортовых комплексов с помощью использования средств языка SysML, позволяющий наглядно и качественно представить строение, структуру и внутреннее взаимодействие ПО БКУ КА. Применение метода позволяет доступными альтернативными средствами представить графическое описание постановки задачи для разработки программно-алгоритмического описания бортового комплекса управления, графическими средствами документировать взаимодействие отдельных элементов любого размера внутри сложных программных комплексов. Также применение предложенных диаграмм в описании алгоритмов БКУ позволяет сократить время анализа и поле возможных событий при возникновении нештатных ситуаций. Недостатком предложенного метода является отсутствие контроля соответствия графического описания ПО соответствующему тексту программ в процессе разработки и отладки ПО БКУ.

Список литературы

1. **Бортовые системы** управления космическими аппаратами: Учеб. пособие / Под ред. А. С. Сырова М.: МАИ-ПРИНТ, 2010. 304 с.
2. **Микрин Е. А.** Бортовые комплексы управления космическими аппаратами и проектирование их программного обеспечения. М.: Изд-во МГТУ им. Баумана, 2003. 336 с.
3. **Космическое** аппаратостроение: Научно-технические исследования и практические разработки ГНПРКЦ "ЦСКБ-Прогресс"/ Под ред. А. Н. Кирилина. Самара: АГНИ, 2011. 280 с.
4. **ГОСТ** 19.001—77 ЕСПД.
5. **Кознов Д. В.** Программная инженерия и визуальное моделирование: воспитание культуры работы с информацией // Программная инженерия. 2015. № 10. С. 3—11.
6. **Иванов Д. Ю., Новиков Ф. А.** Основы моделирования на UML: Учеб. пособие. СПб.: Изд-во Политехн. ун-та, 2010. 249 с.
7. **Friedenthal S., Moore A., Steiner R.** A Practical Guide to SysML: The Systems Modeling Language. Elsevier, Inc., 2009.
8. **Eickhoff J**. Simulating Spacecraft Systems. Springer-Verlag Berlin Heidelberg, 2009.
9. **Калентьев А. А., Тюгашев А. А.** Использование графических языков в жизненном цикле бортового программного обеспечения космических аппаратов // Вестник Самарского государственного аэрокосмического университета. 2010. № 2. С. 248—259.
10. **Коварцев А. Н.** Методы и средства визуального параллельного программирования. Автоматизация программирования: Учеб. Самара: Изд-во Самар. гос. аэрокосм. ун-та, 2011. 168 с.

System Analisis Graphic Instruments Application in Satellites On-Board Attitude and Orbit Control System Software Development

N. V. Ryabogin, n.ryabogin@mokb-mars.ru**, M. A. Shatsky**, msh@mokb-mars.ru, **M. U. Kosinsky**, kosinski@mail.ru, **V. N. Sokolov**, sokolov@mokb-mars.ru, Federal State Unitary Enterprise Moscow experimental design bureau "Mars", Moscow, 127473, Russian Federation, **N. M. Zadorozhnaya**, zanatalie@ya.ru, Bauman Moscow State Technical University**,** Moscow, 105005, Russian Federation

Corresponding author:

Ryabogin Nikolay V., Head of Division, Federal State Unitary Enterprise Moscow experimental design bureau "Mars", Moscow, 127473, Russian Federation, e-mail: n.ryabogin@mokb-mars.ru

Received on May 19, 2016
Accepted on June 02, 2016

This article contains material about system analysis graphic instruments (SAGI) application for satellite on-board attitude and orbit control system (AOCS) software development improvement and technical documentation description improvement. The SAGI application goals are satellites on-board AOCS software quality and reliability increasing, development and modeling improvement. Satellite on-board AOCS is a complex object, which consist of hardware and software components. Exiting methods of technical documentation description are directed for programming specification and exposition and do not allow defining software architecture and logical sequences. The aim is to adapt existing graphical tools for describing complex hierarchical systems for additional graphic description and documentation of AOCS software for observability structure, allowed states and modes of operation, relations between the subsystems and the corresponding simplification of the system debugging. The suggested method of improving the software development via better software architecture describition can be implemented with applying graphical instruments for system analysis. This method is based on Unified Modeling Language (UML) instrument applying in AOCS software development. System Modeling Language (SysML) is also suitable as extended version of UML for development, analysis and verification of complex dynamic systems. Suggested method of AOCS software architecture describing through SAGI allows one to clearly and properly describe architecture, structure and internal sequences of satellite on-board AOCS software. This method suggests several steps, which include whole AOCS software development process. Draft analysis and requirement grouping is first step. AOCS software hierarchic structure design with autonomous function determination is second step. Next step is function modes definition and operation sequences by state diagrams and sequences diagrams. State diagrams based on FMECA can be also applied for AOCS malfunction and failure cases analysis.

Keywords: *system analysis, system analysis graphical instruments, software, satellite on-board control system, satellite*

For citation:

Ryabogin N. V., Shatsky M. A., Kosinsky M. U., Sokolov V. N., Zadorozhnaya N. M. System Analysis Graphic Instruments Application in Satellites On-Board Attitude and Orbit Control System Software Development*, Programmnaya Ingeneria*, 2016, vol. 7, no. 8, pp. 373—382.

DOI: 10.17587/prin.7.373-382

References

1. ***Bortovie*** *sistemy upravleniya kosmicheskimi apparatami. Uchebnoe posobie* (Satellite on-board control systems. Studying materials)/ Ed. A. S. Syrov, Moscow, MAI-PRINT, 2010, 304 p. (in Russian).

2. **Mikrin E. A.** *Bortovie kompleksy upravleniya kosmicheskimi apparatami i proektirovanie ih programmnogo obespecheniya* (Satellite on-board control systems and on-board software development), Moscow, Publishing house of Moscow State Technical University after N. E. Bauman, 2004, 336 p. (in Russian).

3. ***Kosmicheskoe*** *apparatostroenie: nauchno-tehnicheskie issledovaniya I prakticheskie razrabotki* (Space equipment development: Scientific researches and practices in GSLVSDC "CSDC-Progress")/ Ed. A. N. Kirillin, Samara, AGNI, 2011, 280 p. (in Russian).

4. **State Standard** 19.001.

5. **Koznov D. V.** Programmnaya inzheneriya i vizual'noe modelirovanie: vospitanie kul'tury raboty s informaciey (Program Engineering and Visual Modeling: information treating culture training), *Programmnaya Ingeneria* (Program Engineering), 2015, no. 10, pp. 3—11 (in Russian).

6. **Ivanov D. U., Novikov F. A.** *Osnovy modelirovaniya na UML* (Basic practice in UML. Studying materials), Saint-Petesburg, Publishing house of Polytechnic University, 2010, 249 p. (in Russian).

7. **Friedenthal S., Moore A., Steiner R.** *A Practical Guide to SysML: The Systems Modeling Language*, Elsevier, Inc. 2009.

8. **Eickhoff J.** *Simulating Spacecraft Systems*, Berlin Heidelberg, Springer-Verlag, 2009.

9. **Kalent'ev A. A., Tugashov A. A.** Ispolzovanie graficheskich yazikov v zhiznennom cikle bortovogo programmnogo obespecheniya kosmicheskich apparatov (Applying of graphic languages in life cycle of satellite on-board software), *Vestnik Samarskogo Gosudarstvennogo Aerokosmicheskogo Universiteta*, 2010, no. 2, pp. 248—259 (in Russian).

10. **Kovarcev A. N.** *Metody i sredstva vizual'nogo parallelnogo programmirovaniya. Avtomatizaciya programmirovaniya* (Visual Parallel Programming Methods and Instruments. Programming Automatisation), Samara, Publishing house of Samara State Aerospace University, 2011, 168 p. (in Russian).

Уважаемые коллеги!

Приглашаем Вас принять авторское участие в журнале "Программная инженерия".

К опубликованию принимаются статьи, содержание которых соответствует тематике журнала и включает новые результаты исследований, материалы обзорного и методического характера, не опубликованные ранее и не предназначенные к публикации в других печатных или электронных изданиях.

Все статьи проходят обязательное рецензирование в редакции.

Для опубликования статьи в редакцию журнала направляются следующие материалы:

- рукопись статьи в DOC- и PDF-форматах;
- таблицы, иллюстрации и перечень подрисуночных подписей;
- сведения об авторах, содержащие (согласно регламенту РИНЦ) фамилию, имя, отчество, ученые степень и звание, должность, место работы, служебный и домашний адреса, телефоны и E-mail;
- экспертное заключение о возможности публикации статьи в открытой печати;
- англоязычная информация, включающая краткие сведения об авторах, о содержании статьи (расширенная аннотация) и список литературы, которые необходимы для индексирования журнала в международных наукометрических базах данных.

Объем рукописи статьи, предлагаемой к публикации, должен быть не менее 10 и не более 25 страниц машинописного текста, напечатанного на одной стороне белого листа бумаги формата А4 с полями со всех сторон не менее 2 см, с абзацным отступом 1 см, с полуторным межстрочным интервалом, с использованием текстового редактора Microsoft Word (любая версия) с шрифтом Times New Roman размером 14 pt. В указанный объем статьи входят: текст, приложения, иллюстрации, таблицы, список литературы. В отдельных случаях по решению редколлегии объем статьи может быть увеличен. Страницы рукописи должны быть пронумерованы, начиная с первой.

Более подробную информацию о правилах оформления материала см. на сайте http://novtex.ru/prin/rus/for_authors.html

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Е. В. Комиссарова*

Сдано в набор 06.06.2016 г. Подписано в печать 21.07.2016 г.Формат 60×88 1/8.Заказ PI816
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз".Отпечатано в ООО "Адвансед солюшнз".
119071, г.Москва, Ленинский пр-т, д. 19, стр.1. Сайт: **www.aov.ru**